I.E.S.
VENANCIO BLANCO
SALAMANCA

2022

Servidor-cliente
con servicios Open Source

Sustitutivos a servicios “Big tech”

Autor **Manuel Domingo Vergara Carmona**

CICLO FORMATIVO DE GRADO SUPERIOR
Administración de Sistemas Informáticos en Red

Tutores
Sonsoles Rodríguez Hernández
M. Vicenta Tapia Hernández
Carlos Iglesias Blázquez

“La leña que te cortas tú mismo te calienta dos veces”
Henry Ford

Índice de contenidos

1. Resumen **4**

2. Planteamiento inicial. Investigación **5**
2.1. Premisas del Desarrollo de Software Libre 6
2.2. Fases del proyecto y metodología 6
2.3. Datos y preparación de los servidores 8
2.3.1. Servidor de pruebas y de desarrollo 9
2.3.2. Servidor en producción 11
2.4. Aplicaciones y servicios 13
2.4.1. Aplicaciones Big Tech a sustituir 13
2.4.2. Aplicaciones básicas de configuración 14
2.4.2.1. Diferencias entre los dos orquestadores más usados 16
2.4.2.2. Conclusión: Docker Swarm con Portainer 17
2.4.2.2.1. Docker 17
2.4.2.2.2. Docker Swarm 20
2.4.2.2.3. Portainer 21
2.4.2.2.4. Traefik + Let’s encrypt + CloudFlare 22
2.4.3. Otras aplicaciones para la gestión y/o transparencia del proyecto 23
2.4.3.1. Git 23
2.4.3.1.1. Tabla con las diferencias principales entre GitHub y GitLab 23
2.4.3.2. OpenVPN 25
2.4.3.3. Wordpress 25
2.4.4. Aplicaciones descartadas 26

3. Conceptos y definiciones **27**

4. Objetivos **35**
4.1. Primera parte: Servidor en producción, proveedor de servicios Open Source 35
4.2. Posible segunda parte: Proveer a un smartphone con los servicios del servidor 36
4.2.1. ¿Por qué rootear un móvil? 36
4.2.2. Sistemas Operativos Libres para Smartphones 36
4.2.3. Aplicaciones libres para smartphones 36

5. Despliegue del proyecto **37**
5.1. Selección y configuración del servidor VPS 42
5.2. Actualización/Configuración del server. Instalaciones base 43
5.3. Despliegue del proxy inverso y del gestor de orquestación. 45
5.3.1. Traefik 46
5.3.2. Portainer 46
5.3.3. Certificados SSL 47
5.4. Despliegue de aplicaciones 48
5.4.1. Servidor de documentación 50
5.4.2. Servicio VPN 52
5.4.3. Herramienta de métricas 54
5.4.4. CMS 56
5.4.5. Servicio de nube 58
5.4.6. Mail Server 60
5.4.7. Servicio de videollamadas 62
5.4.8. Servicio de chats 64
5.5. Conexión con red perimetral 66

6. Conclusiones **68**
6.1. Sobre el Open Source 69
6.1.1. No es lo mismo “código abierto” que “software libre” 70
6.2. Sobre los microservicios, Docker y los orquestadores de contenedores 71
6.3. Seguridad en la arquitectura 72
6.4. Control de los procesos y distribución de los servicios 73
6.5. Incidencias al desplegar a producción 74

7. Detalles del software utilizado **76**

8. Webgrafía **78**
8.1. Manuales 78
8.2. Comunidades y foros 78
8.3. Guías 78
8.4. Libros 79
8.5. Vídeos 79
8.6. Repositorios de aplicaciones 79
8.7. Otros 79

1. Resumen

Me ha dado mucho que pensar la cita de 2014 del actual director ejecutivo de Apple, Tim Cook: **"Cuando el producto es gratis, el producto eres tú".** En el día a día, por comodidad, **aceptamos servicios de terceros** que suelen ser de los gigantes tecnológicos (Google, Apple, Meta, Amazon, Microsoft...). Estos servicios **se ofrecen como gratuitos** en muchos casos o como freeware.

Tim Cook

Existen otras alternativas entre obtener un servicio gratuito aceptando ser el producto o tener que pagar un servicio de por vida con inevitables vulnerabilidades a nuestra privacidad. Estos servicios se suelen ofrecer como llave en mano: te registras con los datos que requieren y, sin más, ya puedes utilizarlos.

La comodidad del usuario es lo que apremia. Por desconocimiento tecnológico o por falta de tiempo para aprender más sobre servicios concretos, **el usuario acaba aceptando condiciones leoninas** que además ignora: Vivimos en la cultura de no leer los términos y condiciones.

Pues bien, **la alternativa es cubrir las necesidades** de los servicios de un usuario medio, recuperando así el control de sus propios datos **en un servidor propio**. Tiene un **coste económico**, que será el pago del hospedaje y del dominio; y un **coste en conocimientos**, son tecnologías informáticas para profesionales. El servidor **debe ser IaaS, escalable y portable**.

El núcleo del presente proyecto es **instalar todos los servicios necesarios para sustituir los ofrecidos por Big Tech para los smartphones**. Como requisito lógico, las aplicaciones deberán ser **Open Source o software libre**, contando **con una comunidad** que los respalde. Como añadido, para que sea portable **se documentará todo el proceso** para que otros usuarios puedan replicar el proyecto.

Estos usuarios deben tener unos **conocimientos mínimos** de virtualización de SO Linux, de gestión de redes, de microservicios, de orquestación de contenedores de aplicación y, sobre todo, deben ser conscientes de la motivación de esta necesidad. El proceso de instalación/gestión/mantenimiento de servicios propios puede ser tedioso, por eso hay que **tener claros los motivos para anteponerlos a la comodidad/gratuidad** de los mismos servicios ofrecidos por las grandes compañías tecnológicas.

2. Planteamiento inicial. Investigación

Este proyecto está basado en la **arquitectura cliente-servidor,** que es el modelo más extendido en la actualidad. Se centrará en desarrollar una **primera parte** que consiste en preparar un servidor con todos los **servicios que necesita un smartphone** más otros que se entienden **relevantes para su desarrollo y/o despliegue**. También se añadirán servicios que se han considerado necesarios para la transparencia del proyecto en sí.

La segunda parte no entrará dentro de este proyecto más que de manera teórica. Esta parte final consistirá en **formatear un smartphone e instalar un SO libre**, para luego configurar todos los servicios como cliente. **El servidor dispuesto en la primera parte será el proveedor de estos servicios**. Todas las apps deberán ser Open Source o software libre. No se contemplan los servicios ofrecidos por el proveedor de ISP y de telefonía.

En los siguientes puntos se verán las premisas a considerar para el desarrollo del proyecto, las distintas fases y la metodología empleada. Acto seguido, se describirán el diseño y estructura de los servidores de pruebas y de producción. En ambos servidores se empleará la **arquitectura de microservicios**, basada en la arquitectura **orientada a los servicios**.

En el último punto se describirán las aplicaciones y servicios, divididos en: los servicios Big Tech a sustituir, aplicaciones básicas para el correcto funcionamiento del servidor en producción y otras aplicaciones para la gestión y transparencia del proyecto.

Todos estos puntos son el resultado de una **investigación personal, académica y profesional**, que ha tenido una trayectoría de años, con la colaboración de colegas, profesores y compañeros, además de la inestimable aportación de todas esas personas tan importantes que comparten conocimiento en Internet sin ningún ánimo de lucro.

2.1. Premisas del Desarrollo de Software Libre

1. Los buenos trabajos en software comienzan para **solucionar un problema personal**.
2. Los buenos programadores saben qué escribir. **Los grandes saben qué reescribir y reutilizar.**
3. Piensa en **desechar código**: lo terminarás haciendo de todos modos.
4. Con la actitud adecuada **los problemas interesantes te encontrarán**.
5. Cuando un programa deja de interesarte, tu último deber es pasarlo a **un sucesor competente**.
6. Tratar a tus **usuarios como colaboradores** es el camino menos complicado para mejorar.
7. Lánzalo pronto, lánzalo a menudo y **escucha a los usuarios**.
8. En una **comunidad los problemas se identificarán con rapidez** y su solución será obvia para alguien.

Richard Stallman junto a Julian Assange con una foto de Edward Snowden

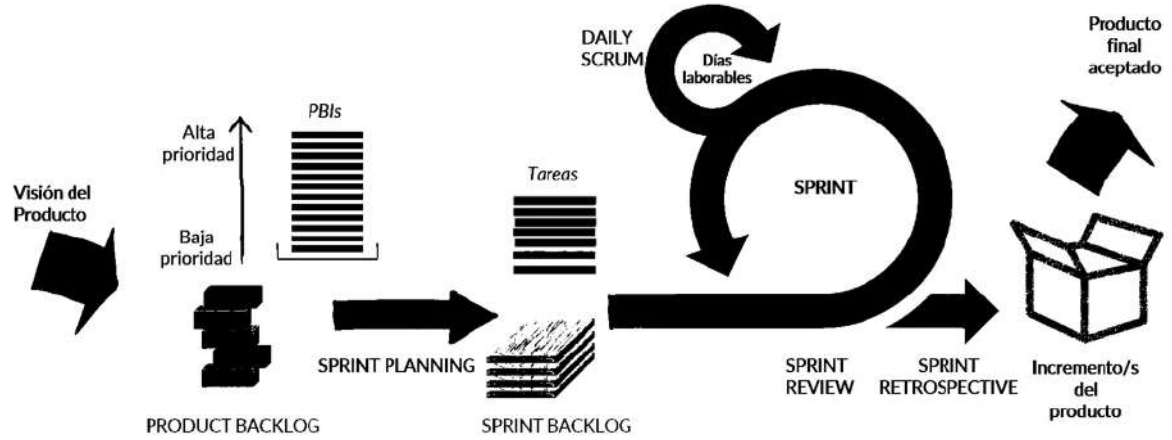
2.2. Fases del proyecto y metodología

El proceso de **resolución de problemas técnicos** se puede resumir en las siguientes fases:

1. Planteamiento e **identificación del problema**.
2. **Investigación**. Búsqueda de soluciones, mejoras y optimización.
3. **Diseño** del conjunto de soluciones a implementar.
4. **Evaluación** de su viabilidad. Idoneidad técnica y funcional. Entorno de pruebas.
5. **Planificación y organización** del trabajo.
6. Desarrollo, construcción, despliegue e **implementación de las soluciones**.
7. **Comprobación**, monitoreo y evaluación.
8. Si es preciso: **Rediseño** (Vuelta al punto 2).

Para el desarrollo del proyecto emplearé dos **metodologías ágiles**: Scrum y Kanban.

Dentro del marco de trabajo **Scrum** estarán divididas las distintas fases del proyecto, con una duración relativa según la fase, por cada uno de los sprints. Con este método de trabajo lo que se pretende es alcanzar el mejor resultado. Las prácticas que se aplican con la metodología Scrum se retroalimentan unas con otras.

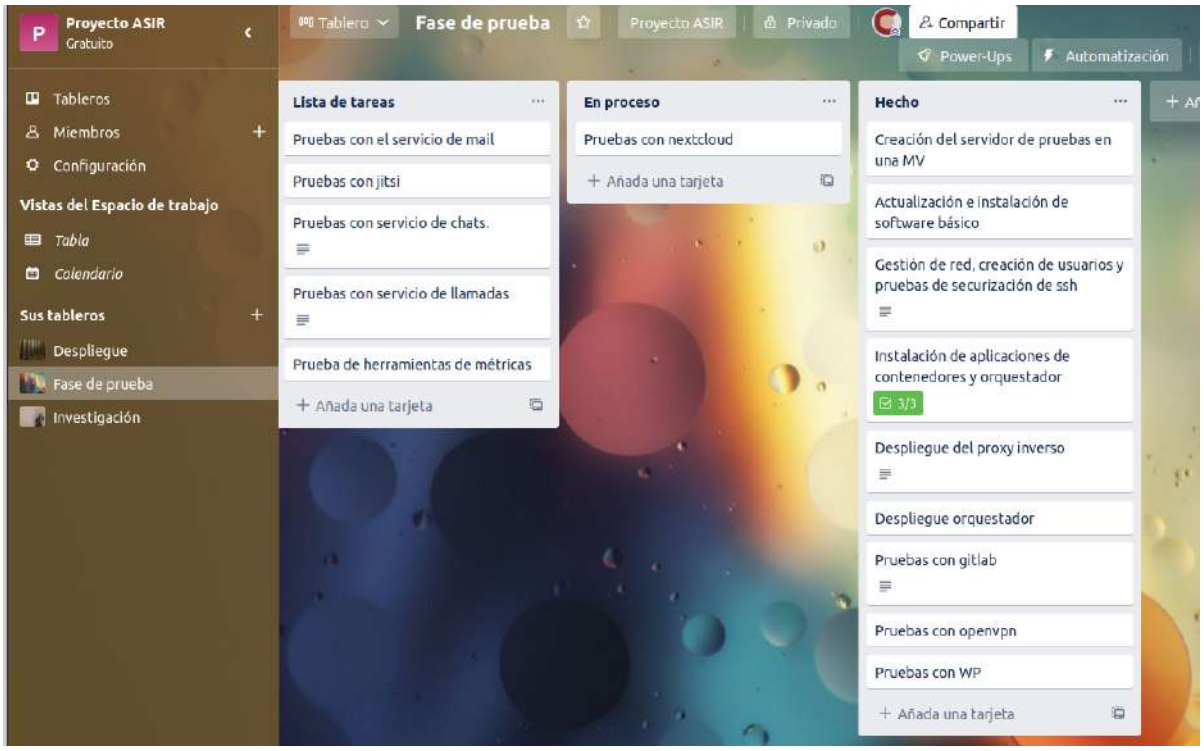


Esquema de Scrum

Por otro lado, tanto para organizar las tareas de investigación, como para las pruebas y la implementación de las distintas herramientas y servicios, se emplea el método de gestión de **tableros Kanban**. Se trata de un método visual de gestión de proyectos que permite visualizar sus flujos de trabajo y la carga de trabajo. El trabajo se muestra organizado por columnas. Normalmente, cada columna representa una etapa del trabajo. Las tareas individuales, representadas por tarjetas visuales, avanzan por las diferentes columnas hasta su finalización.

Como tablero kanban usaré la aplicación web **Trello**: https://trello.com/

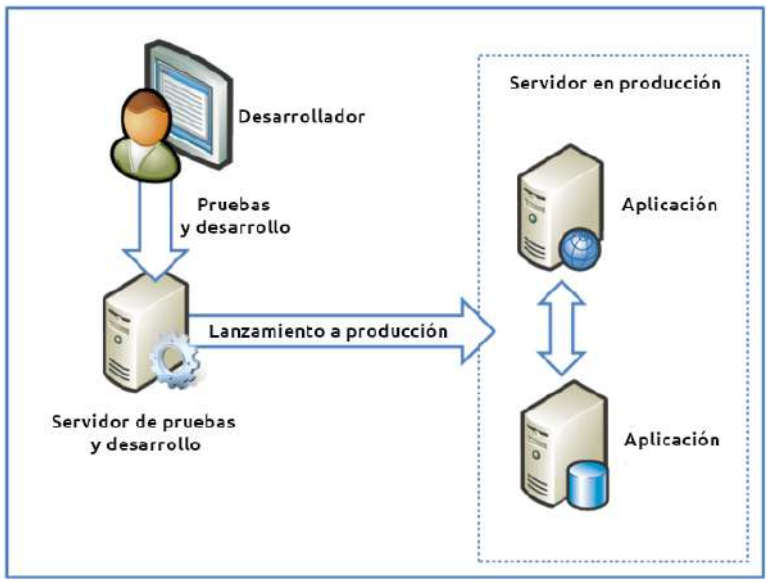
Espacio de trabajo con los tableros en Trello



Tablero de la fase de pruebas

2.3. Datos y preparación de los servidores

Como se ha comentado anteriormente, se preparan **dos servidores**. El primero será en una **MV alojada en un ordenador local** en donde se realizarán pruebas, análisis y, finalmente, el desarrollo de las aplicaciones. El otro servidor será el de **producción** que estará **alojado en un hosting** y tendrá un dominio público para acceder.



Ejemplo de la mecánica de trabajo

En las siguientes subsecciones se darán detalles de ambos servidores, tanto de las características físicas como lógicas.

2.3.1. Servidor de pruebas y de desarrollo

La configuración de este servidor es lo primero que se ha realizado para poner a prueba todos los componentes encontrados en la investigación. Funcionará como **laboratorio** de aprendizaje y los resultados se reflejan en la sección de “Objetivos”. Este servidor también servirá para el **desarrollo que posteriormente se subirá a producción**.

Servidor de desarrollo para pruebas	
Tipo de host	Máquina Virtual en local
Sistema Operativo	Ubuntu 20.04 LTS server (Focal Fossa)
Kernel	GNU/Linux 5.4.0-107-generic x86_64
Disco duro	Formato VDI con almacenamiento dinámico sin cifrar de 50 GB.
Procesador	2 CPUs AMD
Memoria RAM	4108 MB
Acceso de red	Adaptador puente
Dominio	Sin dominio configurado. De todas formas, las aplicaciones están preparadas con el dominio vergaracarmona.es. Para poder acceder se añade al final del archivo hosts del equipo anfitrión la relación IP de la MV con el dominio y subdominios: IP-MV vergaracarmona.es
Servicios nativos (Dentro de la MV) añadidos	OpenSSH, Unzip, zip, Tree, Curl, Ufw
Servicios en contenedores	Docker, Docker-compose, Swarm, Traefik, Portainer

Más datos sobre el hardware utilizado por el servidor con el comando `lscpu`.

```
administrador@ubuntudocker:~$ lscpu
Arquitectura:                    x86_64
modo(s) de operación de las CPUs:    32-bit, 64-bit
Orden de los bytes:              Little Endian
Address sizes:                   48 bits physical, 48 bits virtual
CPU(s):                          2
Lista de la(s) CPU(s) en línea:      0,1
Hilo(s) de procesamiento por núcleo: 1
Núcleo(s) por «socket»:          2
«Socket(s)»                      1
Modo(s) NUMA:                    1
ID de fabricante:                AuthenticAMD
Familia de CPU:                  25
Modelo:                          80
Nombre del modelo:               AMD Ryzen 5 5600H with Radeon Graphics
Revisión:                        0
CPU MHz:                         3293.810
BogoMIPS:                        6587.62
Virtualización:                  AMD-V
Fabricante del hipervisor:       KVM
Tipo de virtualización:          lleno
Caché L1d:                       64 KiB
Caché L1i:                       64 KiB
Caché L2:                        1 MiB
Caché L3:                        16 MiB
CPU(s) del nodo NUMA 0:          0,1
Vulnerability Itlb multihit:     Not affected
Vulnerability L1tf:              Not affected
Vulnerability Mds:               Not affected
Vulnerability Meltdown:          Not affected
Vulnerability Spec store bypass: Vulnerable
Vulnerability Spectre v1:        Mitigation; usercopy/swapgs barriers and __user pointer sanitization
Vulnerability Spectre v2:        Mitigation; LFENCE, STIBP disabled, RSB filling
Vulnerability Srbds:             Not affected
Vulnerability Tsx async abort:   Not affected
Indicadores:                     fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 clflush mmx fxsr sse sse2 ht
                                 syscall nx mmxext fxsr_opt rdtscp lm constant_tsc rep_good nopl nonstop_tsc cpuid extd_apicid tsc_kno
                                 wn_freq pni pclmulqdq ssse3 cx16 sse4_1 sse4_2 x2apic movbe popcnt aes xsave avx rdrand hypervisor la
                                 hf_lm cmp_legacy svm cr8_legacy abm sse4a misalignsse 3dnowprefetch vmmcall fsgsbase avx2 invpcid rds
                                 eed clflushopt arat nrip_save flushbyasid decodeassists
```

Una vez instaladas las aplicaciones anteriores y configurado para empezar a realizar las pruebas, se efectúa en VirtualBox una **instantánea de la MV** para poder recuperar este estado de instalación básica en caso de error fatal.

tuto

Para más detalles consultar el enlace tutorial *“Preparación de entorno de desarrollo local":* https://vergaracarmona.es/preparacion-de-entorno-de-pruebas-local-para-docker/

2.3.2. Servidor en producción

Este servidor **se define en la fase del despliegue**, una vez se conocen los requisitos y necesidades resultantes de la investigación. La selección del hosting tendrá lugar después de una prospección en las promociones que ofrecen las distintas empresas. Se incluyen los detalles en la sección de “Despliegue del proyecto”.

Servidor en producción	
Tipo de host	VPS (Droplet)
Sistema Operativo	Ubuntu 22.04 LTS server (jammy)
Kernel	GNU/Linux 5.15.0-25-generic x86_64
Disco duro	50 GB NVMe SSD
Procesador	1 CPU’s AMD
Memoria RAM	2 GB
Transferencia mensual	2 TB
Dominio principal	vergaracarmona.es
Servicios nativos (Dentro del VPS) añadidos	unzip, zip, tree, curl, ufw
Servicios en contenedores	Docker, Docker-compose, Swarm, Traefik, Portainer

Más datos sobre el hardware utilizado por el servidor con el comando “lscpu”.

```
Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.

root@ubuntu-s-1vcpu-2gb-amd-ams3-01:~# lscpu
Architecture:            x86_64
  CPU op-mode(s):        32-bit, 64-bit
  Address sizes:         40 bits physical, 48 bits virtual
  Byte Order:            Little Endian
CPU(s):                  1
  On-line CPU(s) list:   0
Vendor ID:               AuthenticAMD
  Model name:            DO-Premium-AMD
    CPU family:          23
    Model:               49
    Thread(s) per core:  1
    Core(s) per socket:  1
    Socket(s):           1
    Stepping:            0
    BogoMIPS:            3992.50
    Flags:               fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 clflush mmx fxsr
                          sse sse2 syscall nx mmxext fxsr_opt pdpe1gb rdtscp lm rep_good nopl cpuid extd_apicid t
                         sc_known_freq pni pclmulqdq ssse3 fma cx16 sse4_1 sse4_2 movbe popcnt aes xsave avx f16c
                          rdrand hypervisor lahf_lm svm cr8_legacy abm sse4a misalignsse 3dnowprefetch osvw perfc
                         tr_core ibpb stibp vmmcall fsgsbase bmi1 avx2 smep bmi2 rdseed adx smap clflushopt clwb
                         sha_ni xsaveopt xsavec xgetbv1 xsaves clzero xsaveerptr wbnoinvd arat npt nrip_save umip
                          rdpid
Virtualization features:
  Virtualization:        AMD-V
  Hypervisor vendor:     KVM
  Virtualization type:   full
Caches (sum of all):
  L1d:                   64 KiB (1 instance)
  L1i:                   64 KiB (1 instance)
  L2:                    512 KiB (1 instance)
NUMA:
  NUMA node(s):          1
  NUMA node0 CPU(s):     0
Vulnerabilities:
  Itlb multihit:         Not affected
  L1tf:                  Not affected
  Mds:                   Not affected
  Meltdown:              Not affected
  Spec store bypass:     Vulnerable
  Spectre v1:            Mitigation; usercopy/swapgs barriers and __user pointer sanitization
  Spectre v2:            Mitigation; Retpolines, IBPB conditional, STIBP disabled, RSB filling
  Srbds:                 Not affected
  Tsx async abort:       Not affected
root@ubuntu-s-1vcpu-2gb-amd-ams3-01:~#
```

2.4. Aplicaciones y servicios

En esta sección se enumeran las **aplicaciones usuales de los smartphone** frente a sus posibles servicios y **aplicaciones sustitutas**. También se explican las aplicaciones básicas resultantes de la investigación, así como otras aplicaciones para la transparencia del proyecto y, en la última subsección, las aplicaciones descartadas.

2.4.1. Aplicaciones Big Tech a sustituir

En la siguiente tabla se detallan las **aplicaciones Big Tech** posibles de sustituir, una pequeña descripción, las **aplicaciones del lado del servidor** que ofrecerán los servicios sustitutivos y las **posibles aplicaciones móvil**.

Aplicación	Descripción	Servicio sustitutivo	Aplicación
gmail	Servicio de correo electrónico	Mail server (postfix, dovecot, amavis, spamassassin, ClamAV, OpenDKIM, etc)	K-9 Mail
zoom	Videoconferencias	jitsi server	jitsi meet
Dropbox Calendario Notas Sincronización de contactos Feedly	Almacenamiento en la nube + parte de la suite de google + lector de rss	nextcloud server	nextcloud
whatsapp	Servicio de mensajería instantánea	Matrix	Element
play store	Repositorio de aplicaciones móvil	*F-Droid Server	F-Droid
Google maps	Gestor de mapas, gps	*Open Street Map	IGN app
Last pass	Gestor de contraseña	Keeweb + Webdav	keepass

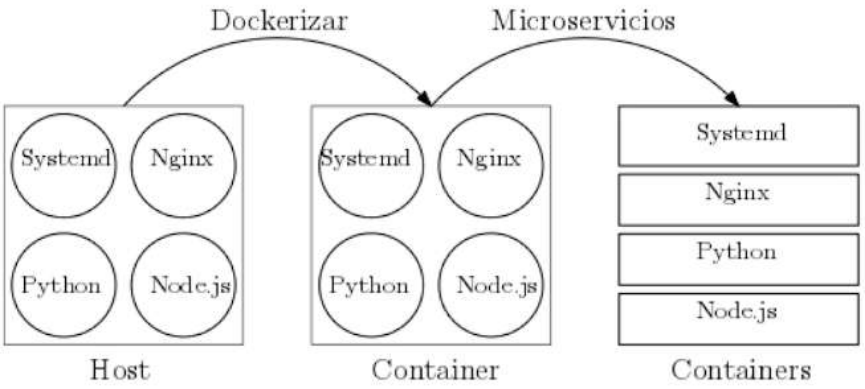
*Servicios OpenSource o software libre que no son Self-Hosting

No todos estos servicios se pueden implementar como “Self-Hosting” y algunas aplicaciones tienen ya servicios open sources mantenidos por comunidades o fundaciones. Los servicios definitivos a implementar se detallan más adelante dentro de la sección de “Objetivos”.

2.4.2. Aplicaciones básicas de configuración

En este punto se describe la **topología lógica** de la comunicación interna del servidor entre las aplicaciones. Será la base del servidor para **la gestión y la seguridad**.

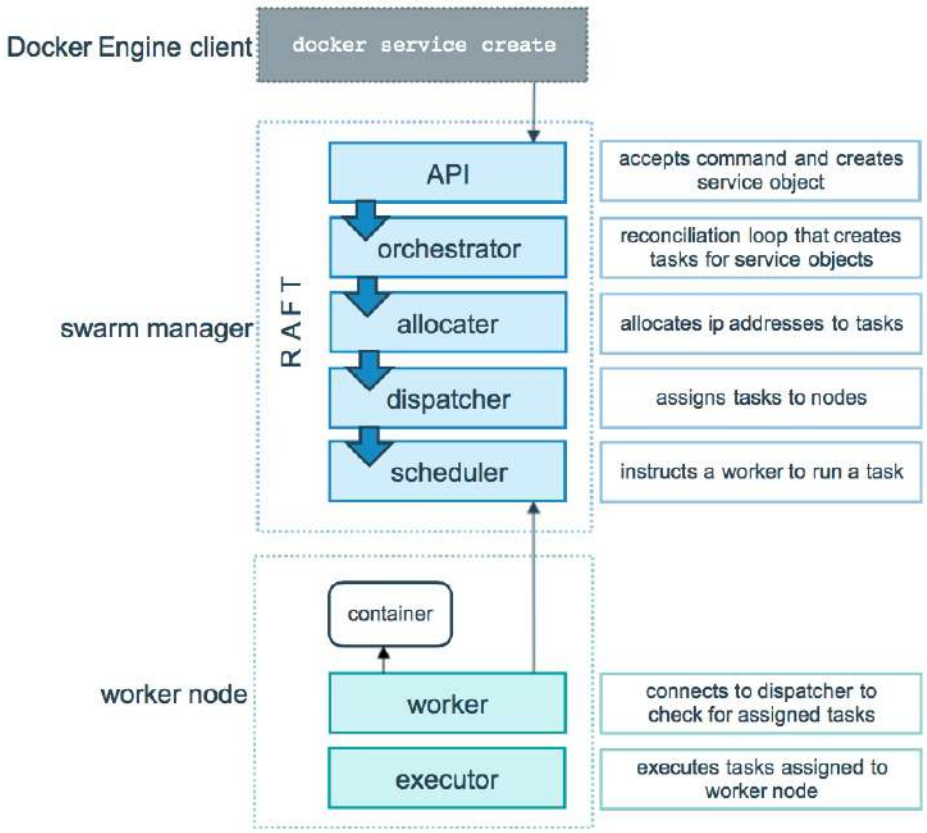
Para aprovechar al máximo los recursos del servidor, los **servicios serán aislados en contenedores de aplicación** con el software **Docker**, estrechamente relacionados con las arquitectura de microservicios.



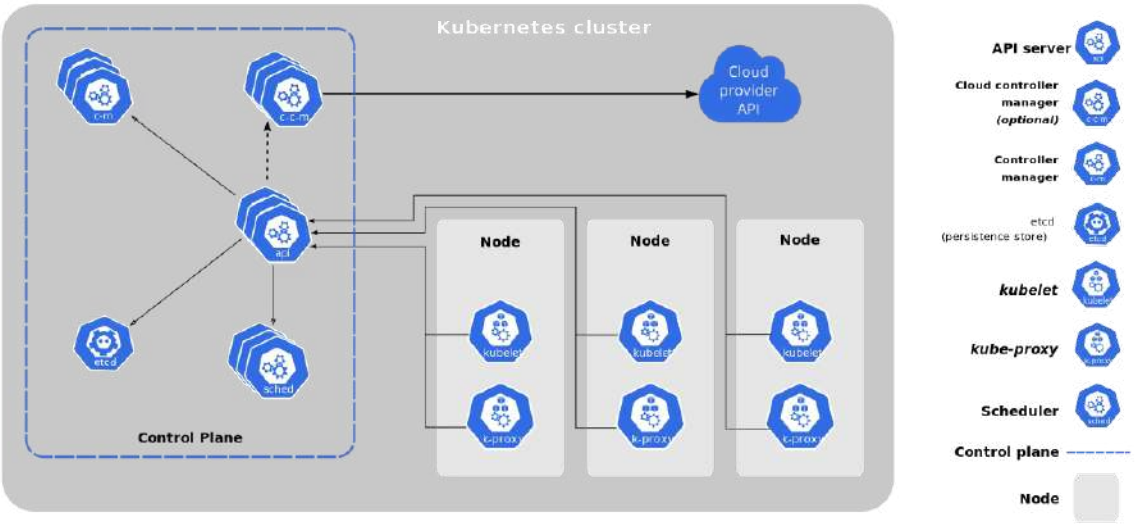
Una pila de servicios pueden estar dockerizados sin ser microservicios, en un contenedor monolítico.

Para la **orquestación de los contenedores** ha habido una intensa búsqueda, así como diferentes debates entre compañeros y colegas. **Existen muchas herramientas**, sin entrar en detalles, **enumero tres posibilidades** junto un diagrama de su funcionamiento:

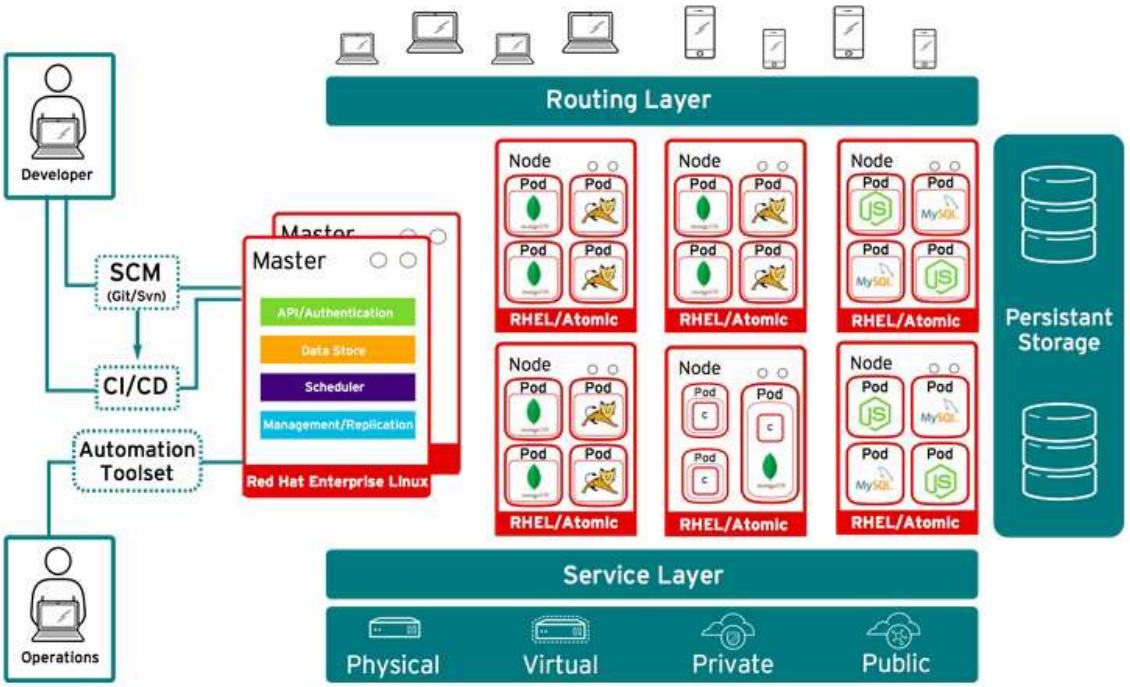
- **Docker Swarm**



- **Kubernetes**



- **Red Hat Openshift**



Ordenadas **de menor complejidad a nivel overkill** junto a posibles clientes gŕaficos:

- Docker Swarm + Portainer (Cliente gráfico).
- Kubernetes + cliente gráfico (Kubernetes Dashboard, Lens, Octant o kubenav).
- Openshift. Incluye cliente gráfico, Docker y kubernetes.

2.4.2.1. Diferencias entre los dos orquestadores más usados

Características	Kubernetes	Docker Swarm
Instalación	Compleja	Simple
Balanceador de carga	Se requiere la intervención manual para el equilibrio de carga	Automático
Escalabilidad	El escalado y el despliegue son comparativamente más lentos	Los contenedores se despliegan mucho más rápido
Cluster	Difícil de configurar	Fácil de configurar
Configuración del contenedor	Los comandos YAML deben reescribirse al cambiar de plataforma	Un contenedor puede desplegarse fácilmente en diferentes plataformas
Registro y monitorización	Consta de herramientas integradas para gestionar ambos procesos	No se necesitan herramientas para el registro y la supervisión
Disponibilidad	Alta disponibilidad cuando los pods se distribuyen entre los nodos	Aumenta la disponibilidad de las aplicaciones gracias a la redundancia
Volúmenes de datos	Compartidos con contenedores del mismo pod	Pueden ser compartidos con cualquier contenedor

2.4.2.2. Conclusión: Docker Swarm con Portainer

Kubernetes está más destinado a gestionar clusters repartidos en distintas ubicaciones. Openshift es capaz de manejar clusters de entornos muy diversos. Al no tener estas necesidades **he optado por la sencillez: Docker swarm** con el cliente gráfico **Portainer** con contenedores monolíticos y en algún caso con algún microservicio aislado.

En los siguientes puntos se explican los detalles de las aplicaciones básicas citadas junto con otras que formarán la seguridad del servidor con el exterior. Los conceptos más destacados se explican más adelante en la sección de “Conceptos y definiciones”.

2.4.2.2.1. Docker

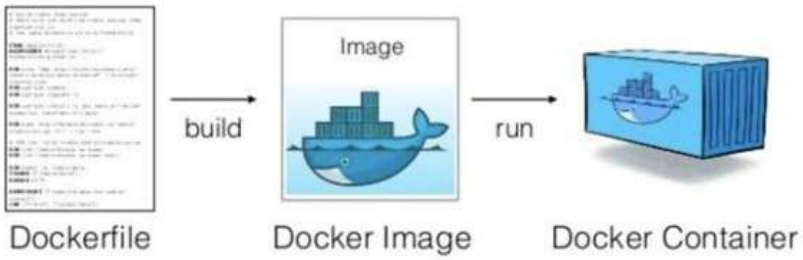
El término "**Docker**" se aplica a diferentes conceptos, entre los que se incluyen un **proyecto de la comunidad open source** y sus herramientas; **Docker Inc.**, la principal empresa promotora del proyecto; y las **herramientas** que respaldan tanto la comunidad como la empresa formalmente. Como **la tecnología y la compañía comparten el mismo nombre puede dar pie a confusión**.

Aspectos importantes:

- El sistema de software de TI llamado "Docker" es la **tecnología de organización en contenedores** que posibilita la creación y el uso de los contenedores de Linux.
- La **comunidad open source** Docker se encarga de mejorar estas tecnologías para beneficiar a todos los usuarios.
- La empresa **Docker Inc. se basa en el trabajo de la comunidad** Docker para aumentar la seguridad de las herramientas y comparte los avances con el resto de la comunidad. Entonces, brinda **soporte** a las tecnologías mejoradas y reforzadas para los clientes empresariales.

En el proyecto **se empleará la versión docker de la comunidad “Community edition“.**

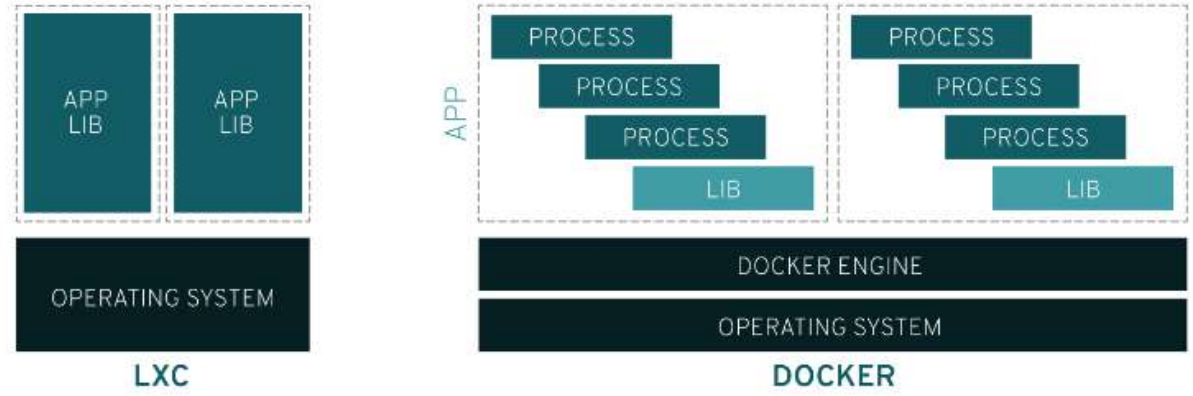
La tecnología Docker utiliza el **kernel de Linux** y sus funciones, como los **cgroups** y los **namespaces**, para dividir los procesos y ejecutarlos de manera independiente. Es el propósito principal de los contenedores, **ejecutar procesos y aplicaciones por separado** para aprovechar la infraestructura al máximo y, al mismo tiempo, conservar la seguridad que se obtendría con los sistemas individuales.



Las herramientas de contenedores, como Docker, tienen un modelo de implementación **basado en imágenes**, que permite compartir una aplicación o un conjunto de servicios con todas sus dependencias en varios entornos. Docker también **automatiza la implementación** de las aplicaciones (o los conjuntos de procesos que las constituyen) en el entorno de contenedores.

Estas herramientas están **diseñadas a partir de los contenedores de Linux**, por eso la tecnología Docker es sencilla y única. Además, ofrecen a los usuarios **acceso sin precedentes** a las aplicaciones, la posibilidad de realizar **implementaciones en poco tiempo** y el **control sobre las versiones** y su distribución.

Hay que diferenciar entre Docker y los contenedores Linux, conceptos que suelen confundirse. Originalmente, **Docker se diseñó a partir de la tecnología LXC**, la cual mucha gente asocia con los contenedores "tradicionales" de Linux, pero desde aquel entonces se ha ido alejando de esa dependencia. **LXC funcionaba como una virtualización** ligera, pero no ofrecía una buena experiencia al desarrollador ni al usuario. La tecnología Docker no solo ofrece la capacidad para ejecutar los contenedores, sino que también **facilita su creación y diseño**, así como **el envío y el control de versiones** de las imágenes, entre otras funciones.



Gráficos con las diferencias entre LXC y Docker

Los contenedores tradicionales de Linux usan un sistema init para gestionar varios procesos, lo cual permite que las aplicaciones completas se ejecuten como una sola. La tecnología **Docker pretende que se dividan las aplicaciones en sus procesos** individuales y ofrece las herramientas para hacerlo. Este enfoque de separación de los elementos tiene sus ventajas:

- Modularidad
- Capas y control de versiones de imágenes
- Restauración
- Implementación rápida

Los **componentes principales** de Docker son:

- **Docker Client** - Cliente en línea de comandos (CLI) que gestiona Docker Engine en local o en remoto.
- **Docker Engine** - Demonio que corre sobre distribución Linux anfitriona y expone una API externa para la gestión de contenedores e imágenes
- **Docker Registry** - Almacena las imágenes generadas por Docker Engine. Podemos distribuir nuestras imágenes o tenerlas en un registro privado. Se pueden utilizar Docker Hub, gitlab, cassandra, etc.

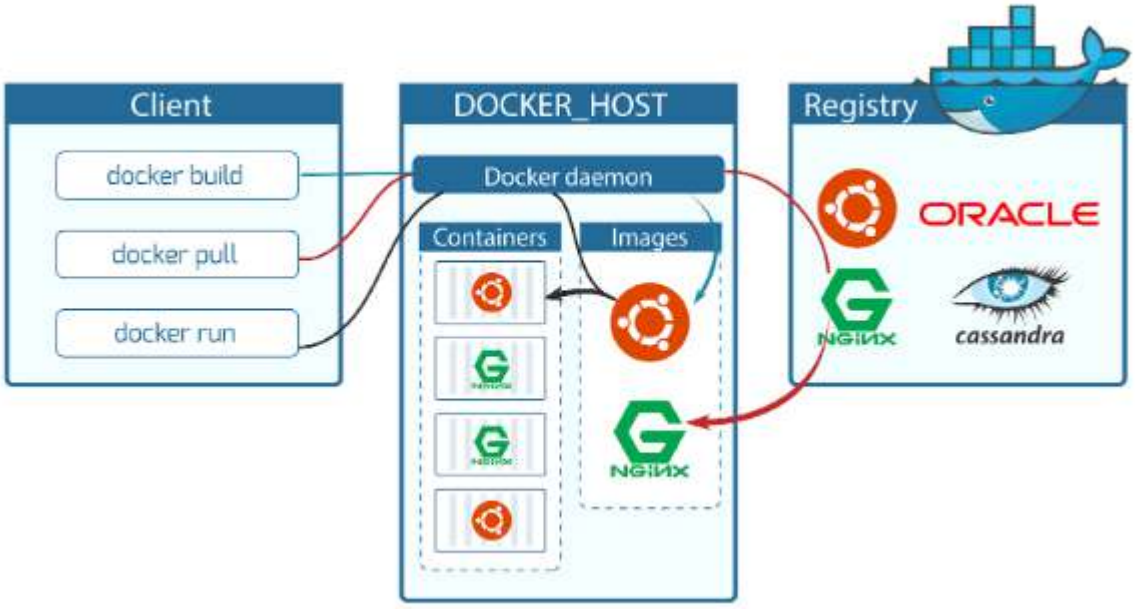


Gráfico con la interacción de los componentes

Más información:

- Uso básico de Docker https://vergaracarmona.es/uso-basico-de-docker/
- Conseguir imágenes de Docker Hub https://vergaracarmona.es/conseguir-imagenes-en-docker-hub-desde-la-terminal/

2.4.2.2.2. Docker Swarm

Docker Swarm permite distribuir aplicaciones en la red a modo de tareas de forma rápida y cómoda y ahorrando recursos, es el administrador de clústeres de Docker que con la denominación de **"Swarm Mode"** se incluye como función nativa de la Docker engine desde la versión 1.12.0 y con ello forma parte del software nuclear de la plataforma.

Docker Swarm permite **escalar las aplicaciones** basadas en contenedores, gestionándolas en tantas instancias y en tantos nodos de red como se requiera. Por el contrario, para ejecutar en un clúster una aplicación multicontenedor, conocida también como **"stack"** (lote), es mejor recurrir a la herramienta Docker Compose.

El cliente gráfico a implementar para la gestión de Docker Swarm será **Portainer**, el cuál será uno de los primeros contenedores que se correrá, junto con **Traefik**, para tener los **contenedores básicos** para la gestión y securización de las aplicaciones.

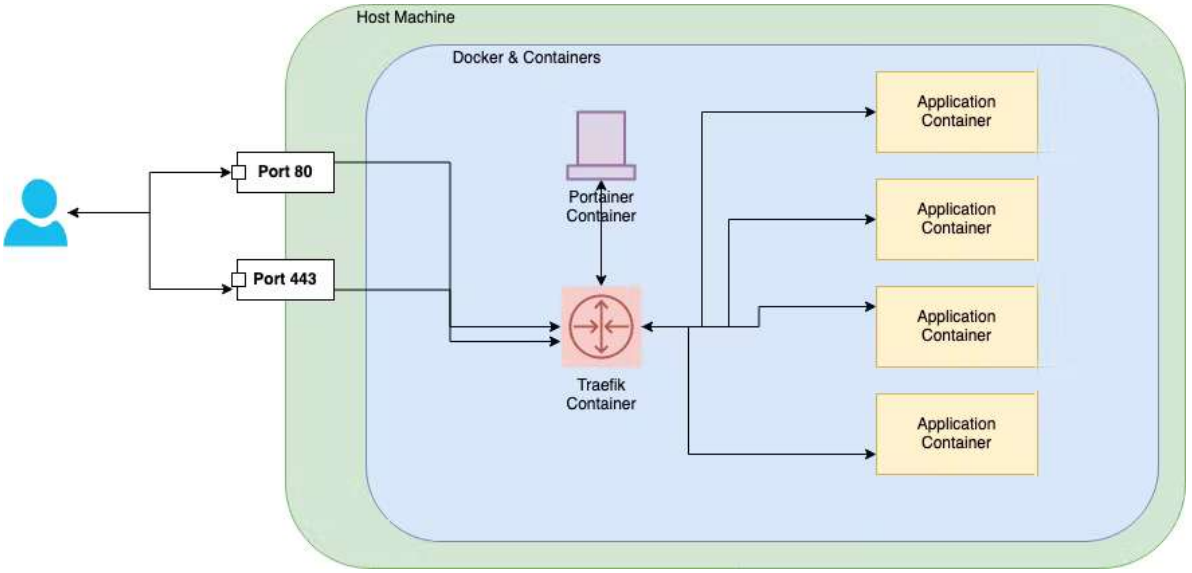
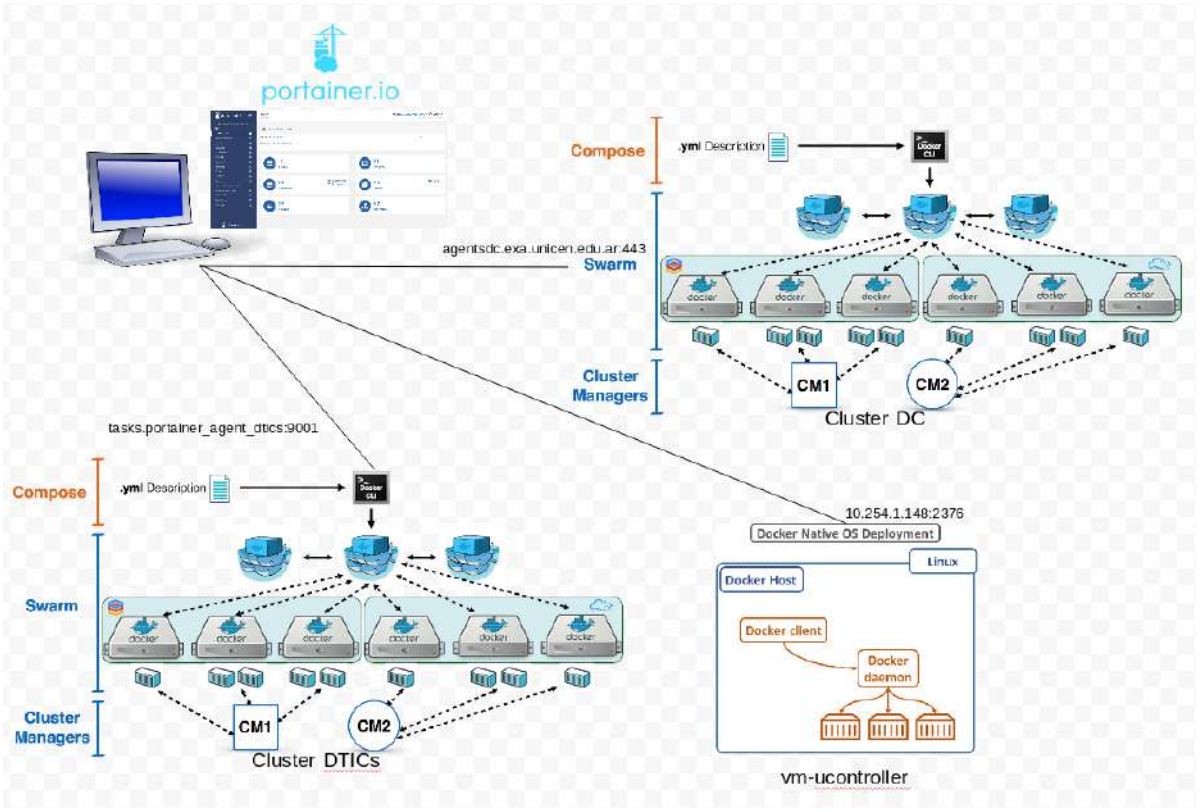


Diagrama de flujo entre el usuario y las aplicaciones

2.4.2.2.3. Portainer

Portainer proporciona una interfaz de usuario excelente y altamente informativa para **administrar todos los aspectos de Docker**. Los gráficos y los paneles proporcionan gráficos **en tiempo real** del uso de los recursos del contenedor. Puede iniciar y administrar contenedores y pilas con unos pocos clics del mouse, entre muchas otras ventajas.



Ejemplo de diagrama swarm con portainer en cliente para gestionar clusters y MV

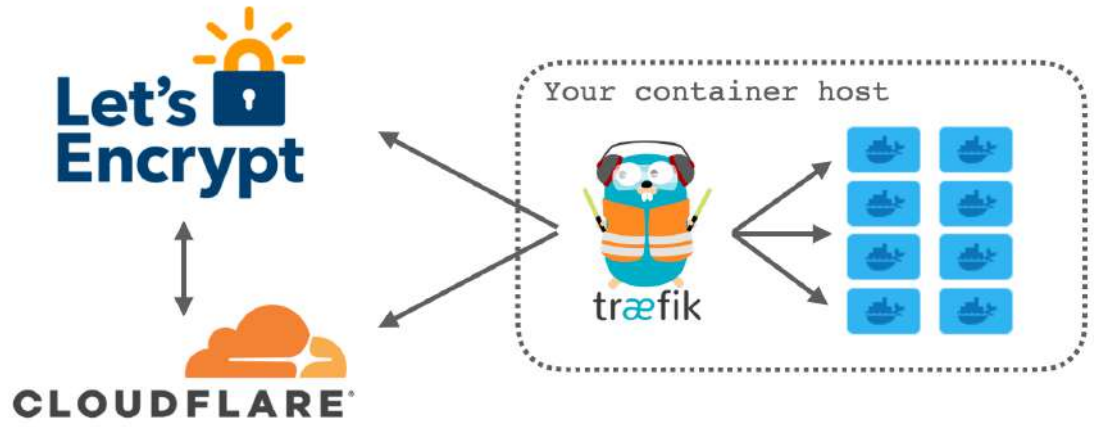
2.4.2.2.4. Traefik + Let's encrypt + CloudFlare

Traefik es un **proxy inverso y balanceador de carga** adaptado a la computación en la nube mediante **microservicios**. Se integra con los principales componentes de infraestructura configurándose a sí mismo automáticamente y de forma dinámica. Es simple de operar pero c**apaz de manejar sistemas complejos** y grandes en entornos diversos y diferentes capas de la pila de red como HTTP, TCP o UDP.

Proporciona funcionalidades de intermediario aumentando sus capacidades para realizar balanceo de carga, este proxy actúa en diferentes capas de la pila de red. Otras funcionalidades son la **limitación de peticiones, la duplicación o mirroring, circuit breaker y la autenticación.**

También soporta terminación de **SSL** y puede usar un **proveedor ACME** como **Let's Encrypt** para la generación automática de certificados. Además, se integra con herramientas para la observabilidad como **Prometheus** para métricas y **Zipkin** y/o **Elastic** para trazabilidad y registro de trazas.

Para más seguridad, el servidor se conectará con una **red perimetral** llamada **CloudFlare**. Es un sistema gratuito que actúa como un proxy (intermediario) entre los visitantes del sitio y el servidor. Al actuar como un proxy, CloudFlare **guarda temporalmente contenido estático** del sitio, el cual disminuye el número de peticiones al servidor pero sigue permitiendo a los visitantes el acceso al sitio.

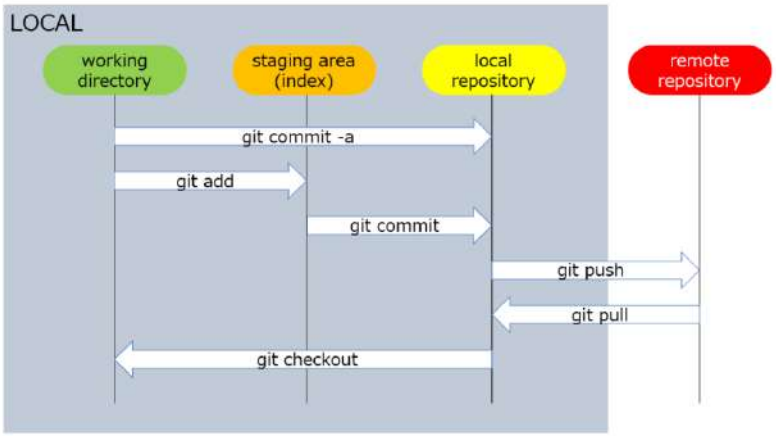


Seguridad lógica

2.4.3. Otras aplicaciones para la gestión y/o transparencia del proyecto

2.4.3.1. Git

Git es un **software de control de versiones** pensando en la eficiencia, la confiabilidad y compatibilidad del mantenimiento de versiones de aplicaciones cuando tienen un gran número de archivos de código fuente. Su propósito es llevar **registro de los cambios** en archivos de computadora incluyendo **coordinar el trabajo** que varias personas realizan sobre archivos compartidos en un **repositorio de código**.



Ciclo básico de trabajo en Git

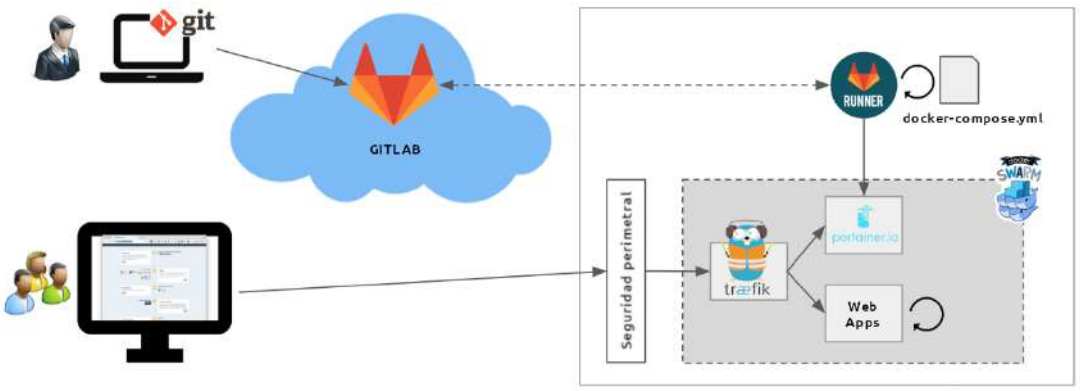
Para el sistema de control de versiones se pueden utilizar distintos servidores de repositorios entre los que destacan **GitLab y GitHub**. Al margen de la similitud principal de su base común en Git, hay algunas diferencias destacables.

2.4.3.1.1. Tabla con las diferencias principales entre GitHub y GitLab

GitHub	GitLab
Los elementos se pueden rastrear en varios repositorios	Los elementos no se pueden rastrear en varios repositorios
Repositorios privados exigen versión de pago	Repositorios privados en versión gratuita
No hay opción gratuita self-hosting	Opción gratuita de self-host
Integración continua solo mediante herramientas de terceros como Travis CI, CircleCI etc.	Integración continua gratuita incluida
No cuenta con plataforma de implementación integrada	Implementación de software de Kubernetes
Rastreo completo de comentarios	Sin rastreo de comentarios
No hay opción de exportación de elementos como archivo CSV	Opción de exportación de elementos como archivo CSV por correo electrónico
Panel personal para rastrear elementos y solicitudes pull	Panel de análisis para planificar y supervisar proyectos

Las empresas que hay detrás de estas herramientas hace que me decante por usar GitLab. GitLab lo gestiona **GitLab Inc**. y GitHub, aunque sea la plataforma más usada, ahora mismo es una filial de **Microsoft** desde que la compró en 2018. No obstante, GitLab es pesada por todas las funcionalidades que contienen, con lo que el software concreto que a implementar en el proyecto será **Gitea**, que es un folk de GitLab.

Explicando un poco más de **GitLab,** comenzó como una plataforma de código abierto self-hosting bajo una licencia MIT. Todavía conserva esta versión libre: GitLab CE (Community Edition). Pero hay que diferenciarla de la versión GitLab EE (Enterprise Edition), que Gitlab Inc comenzó a desarrollar bajo una licencia privativa en febrero de 2014. Esta versión contiene características que no están presentes en la versión libre.



Canalización de CI/CD

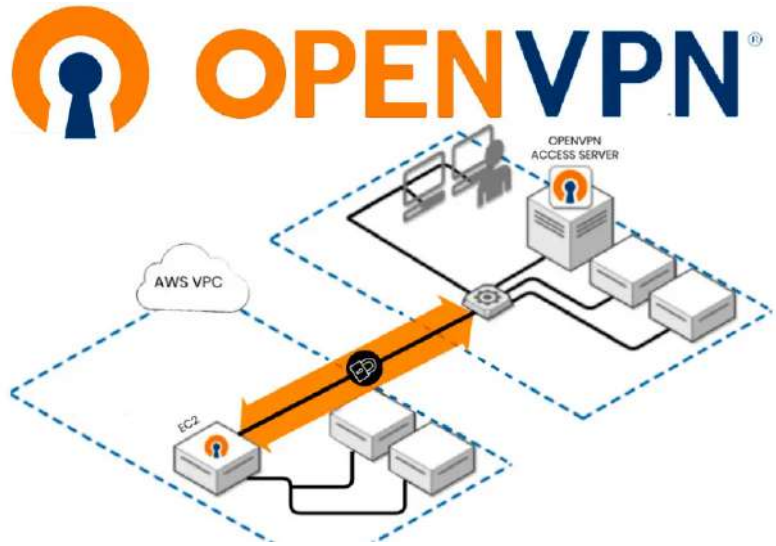
En definitiva, **GitLab** es una suite completa que permite gestionar, administrar, crear y conectar los repositorios con diferentes aplicaciones y hacer todo tipo de integraciones con ellas, ofreciendo un ambiente y una plataforma en la cual se puede **registrar las etapas de desarrollo y DevOps** de un proyecto.

Por otra parte, explicando un poco sobre **Gitea,** es un paquete de software de código abierto que también soporta el auto-alojamiento y proporciona una primera instancia pública gratuita. Este es un fork de **Gogs, un clon de GitLab**. Aunque **Gogs** era un proyecto de código abierto, su repositorio estaba bajo el **control exclusivo** de un único mantenedor, lo que limitaba la cantidad de información y la velocidad con la que la comunidad podía influir en el desarrollo. Frustrados por esto, la comunidad de desarrolladores crearon Gitea.

Más información: “Instalar Git en Ubuntu” https://vergaracarmona.es/instalar-git-en-ubuntu/

2.4.3.2. OpenVPN

OpenVPN es una herramienta de conectividad basada en software libre: SSL, VPN Virtual Private Network. OpenVPN ofrece conectividad punto-a-punto con validación jerárquica de usuarios y host conectados remotamente. Resulta una muy buena opción en tecnologías Wi-Fi (redes inalámbricas IEEE 802.11) y soporta una amplia configuración, entre ellas balanceo de cargas.



Figuración gráfica de una VPN

2.4.3.3. Wordpress

WordPress es un sistema de gestión de contenidos web (CMS), un sistema para publicar contenido en la web de forma sencilla. Es el líder absoluto a nivel mundial para la creación de webs desde hace muchos años.

© W3Techs.com	usage	change since 1 March 2022	market share	change since 1 March 2022
1. WordPress	43.0%	-0.3%	64.4%	-0.7%
2. Shopify	4.4%		6.5%	-0.2%
3. Wix	2.2%	+0.2%	3.3%	+0.3%
4. Squarespace	2.0%	+0.2%	2.9%	+0.2%
5. Joomla	1.7%		2.5%	

percentages of sites

Estadística de los CMS más usados de https://w3techs.com/

2.4.4. Aplicaciones descartadas

En la búsqueda de posibles aplicaciones y servicios a implementar **se han descartado** algunas por su **complejidad, por su diseño o funcionamiento herrumbroso** y/o, simplemente, por considerar que **no tienen cabida** en este proyecto concreto. Aun así, siendo de interés para los posibles microservicios que proveen a un smartphone, se ha creado la siguiente lista mencionando las más relevantes:

- **Mastodon** - Red social libre y federada. Se diferencia de Twitter en ser una federación descentralizada de servidores ejecutando software de código abierto. Su código y documentación está disponible en el repositorio github. Eso significa que los usuarios están esparcidos en diferentes comunidades autónomas e independientes llamadas "instancias" (servidores) cuya red se llama fediverso pero aun así unificados por medio de la posibilidad de interactuar entre sí.
- **Keeweb + Webdav** - Keeweb es un gestor de contraseñas gratuito y de código abierto compatible con KeePass, disponible como versión web y aplicación de escritorio. WebDAV es un protocolo que nos permite guardar archivos, editarlos, moverlos y compartirlos en un servidor web.
- **Privacyidea** - es un servidor de autenticación modular que se utiliza para mejorar la seguridad de aplicaciones con autenticación de dos factores. Originalmente se utilizaba para la autenticación OTP (One Time Password) pero en la actualidad incorpora otros servicios como la respuesta al desafío, U2F, Yubikeys, claves SSH y certificados x509.
- **BigBlueButton** es un sistema de conferencia web de código abierto. Está basado en el sistema operativo GNU/ Linux y se ejecuta en Ubuntu. Además de varios servicios de conferencia web, tiene integraciones para muchos de los principales sistemas de aprendizaje y gestión de contenido. Mi intención era usarlo para llamadas de voz, ya que es muy estable para usar desde un smartphone con poca cobertura, pero los requisitos que necesita para el servidor son:
    - 16 GB de memoria con swap habilitado
    - 8 núcleos de CPU, con alto rendimiento de un solo hilo
    - 500 GB de espacio libre en disco (o más) para las grabaciones, o 50 GB si la grabación de sesiones está desactivada en el servidor.
- **CAS** para un servidor de autenticación SSO. El Servicio de Autenticación Central (CAS) es un protocolo de inicio de sesión único para la web cuyo objetivo es permitir que un usuario acceda a varias aplicaciones proporcionando sus credenciales (como el ID de usuario y la contraseña) una sola vez. También permite a las aplicaciones web autenticar a los usuarios sin tener acceso a sus credenciales de seguridad, como la contraseña.

3. Conceptos y definiciones

A lo largo del documento, se utilizan **palabras técnicas, definiciones y conceptos** que se explican en esta sección. Salvo error u omisión, se respeta el orden de aparición.

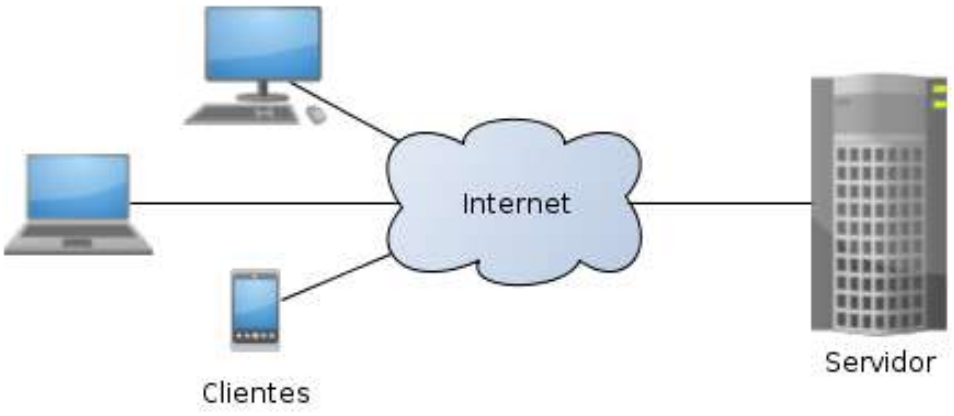
Big Tech - Gigantes tecnológicos también conocidas como Gigantes Tech, Cinco Grandes, o GAFAM son las mayores empresas en tecnología de la información de la industria como Google, Apple, Meta, Amazon o Microsoft.

Metodología ágil - Envuelve un enfoque para la toma de decisiones en los proyectos de software, que se refiere a métodos de ingeniería del software basados en el desarrollo iterativo e incremental, donde los requisitos y soluciones evolucionan con el tiempo según la necesidad del proyecto.

Scrum - Es un proceso en el que se aplican de manera regular un conjunto de buenas prácticas para trabajar colaborativamente, en equipo, y obtener el mejor resultado posible de un proyecto. Se realizan entregas parciales y regulares del producto final. Especialmente indicado para proyectos en entornos complejos, donde se necesita obtener resultados pronto, donde los requisitos son cambiantes o poco definidos, donde la innovación, la competitividad, la flexibilidad y la productividad son fundamentales.

Kanban - Método para gestionar el trabajo intelectual, con énfasis en la entrega justo a tiempo, mientras no se sobrecarguen los miembros del equipo. En este enfoque, el proceso, desde la definición de una tarea hasta su entrega al cliente, se muestra para que los participantes lo vean y los miembros del equipo tomen el trabajo de una cola.

Arquitectura cliente-servidor - es un modelo de diseño de software en el que las tareas se reparten entre los proveedores de recursos o servicios, llamados servidores, y los demandantes, llamados clientes.



Ejemplo de arquitectura cliente-servidor

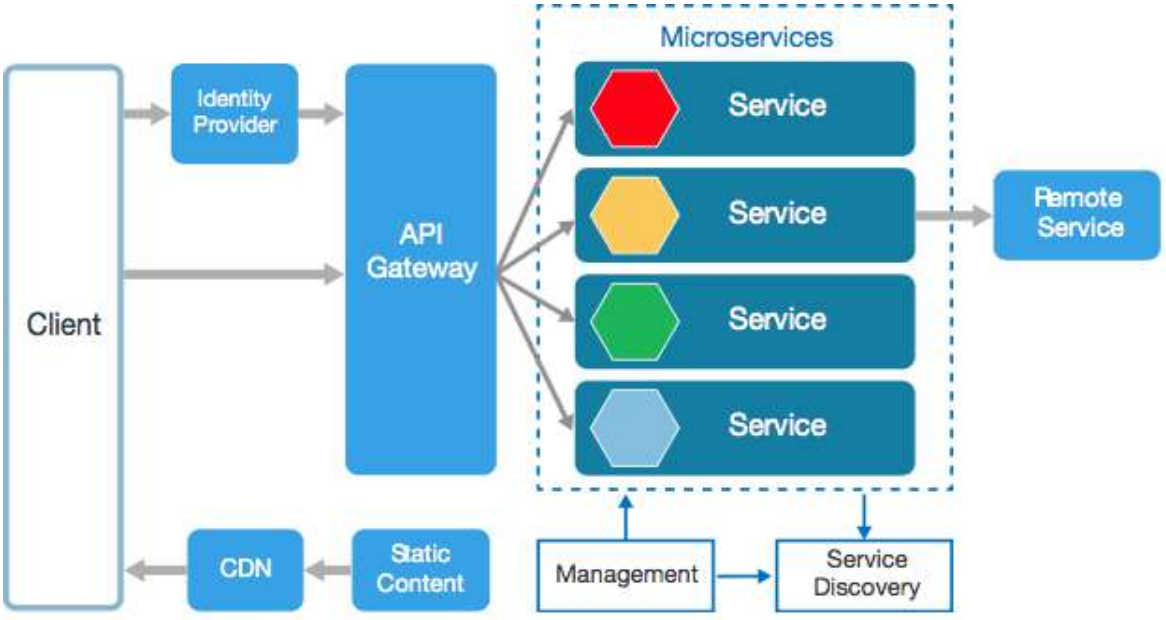
SO - Un sistema operativo es el conjunto de programas de un sistema informático que gestiona los recursos de hardware y provee servicios a los programas de aplicación de software.

Servicios http/https - Protocolo de transferencia de hipertexto. Es el protocolo de comunicación que permite las transferencias de información a través de archivos (XHML, HTML…) en la World Wide Web.

MV - Una máquina virtual es un software que simula un sistema de computación y puede ejecutar programas como si fuese una computadora real. Este software en un principio fue definido como "un duplicado eficiente y aislado de una máquina física".

Self-Hosting - Es el uso de un programa de computadora como parte de la cadena de herramientas o SO que produce nuevas versiones de ese mismo programa, por ejemplo, un compilador que puede compilar su propio código fuente. El software self-hosting es común en computadoras personales, en servidores y en sistemas más grandes. Otros programas que suelen ser self-hosting incluyen kernels, ensambladores, intérpretes de línea de comandos y software de control de versiones. En resumen, es la práctica de alojar y gestionar aplicaciones en un equipo propio, sin control ajeno, en lugar de consumir de proveedores de SaaS.

MSA - Los microservicios son uno de los métodos de desarrollo de software. Una arquitectura de microservicios es un método para desarrollar aplicaciones de software como un conjunto de servicios modulares desplegables de forma independiente, en los que cada servicio ejecuta un proceso único y se comunica a través de un mecanismo claro y bien definido para servir a un objetivo del negocio. Se suele considerar como una forma específica de realizar una arquitectura orientada a servicios.



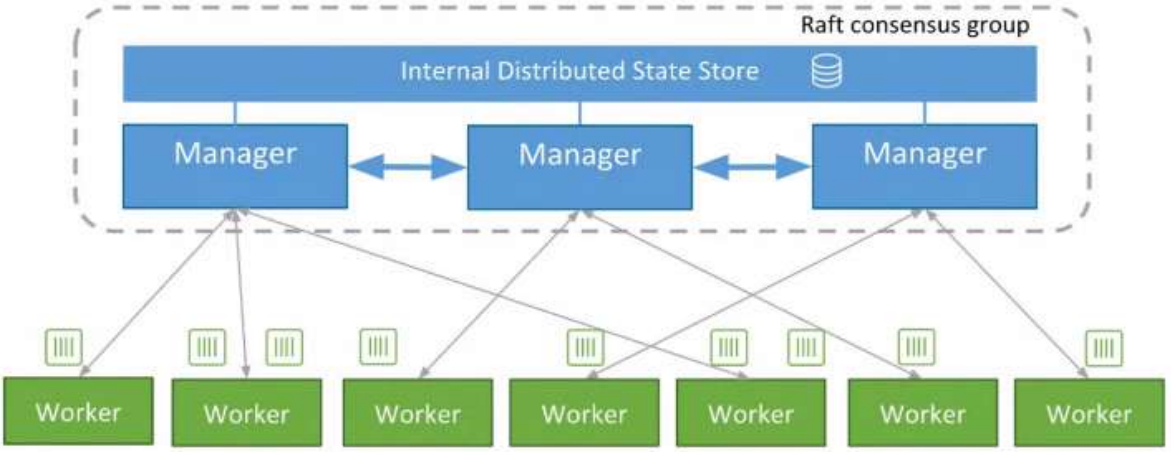
Ejemplo de un diseño de arquitectura de microservicios

SOA - La arquitectura orientada a servicios es una arquitectura de software en base a la que los distintos componentes de la aplicación se relacionan entre sí, y proporcionan servicios al resto de componentes mediante un protocolo de comunicaciones en red. Puede coordinar o poner en comunicación dos o más servicios.

API - La Interfaz de Programación de Aplicaciones es un conjunto de definiciones y protocolos que se utiliza para desarrollar e integrar el software de las aplicaciones, permitiendo la comunicación entre dos aplicaciones de software a través de un conjunto de reglas.

Contenedor de aplicación - La contenerización de aplicaciones es un método de virtualización de nivel de SO para implementar y ejecutar aplicaciones distribuidas sin lanzar una MV completa para cada aplicación. En su lugar, varios sistemas aislados se ejecutan en un único host de control y acceden a un único kernel. Los contenedores de aplicaciones contienen los componentes, como archivos, variables de entorno y bibliotecas, necesarios para ejecutar el software deseado. Debido a que los recursos se comparten de esta manera, se pueden crear contenedores de aplicaciones que ponen menos presión a los recursos globales disponibles. Se confunde los contenedores con el término MV, el cuál no se ajusta del todo ya que los contenedores aislan más bien los procesos sin necesidad de replicar recursos como el kernel que aprovecha del anfitrión.

Orquestador de contenedores - Son herramientas que dirigen el comportamiento de los contenedores pudiendo automatizar el despliegue, la gestión y el escalado de las aplicaciones basadas en contenedores. Estas herramientas son necesarias en entornos en los que tenemos que manejar un sistema con muchos contenedores, que dan distintos servicios (base de datos, servidor web, métricas, la propia aplicación, ...)



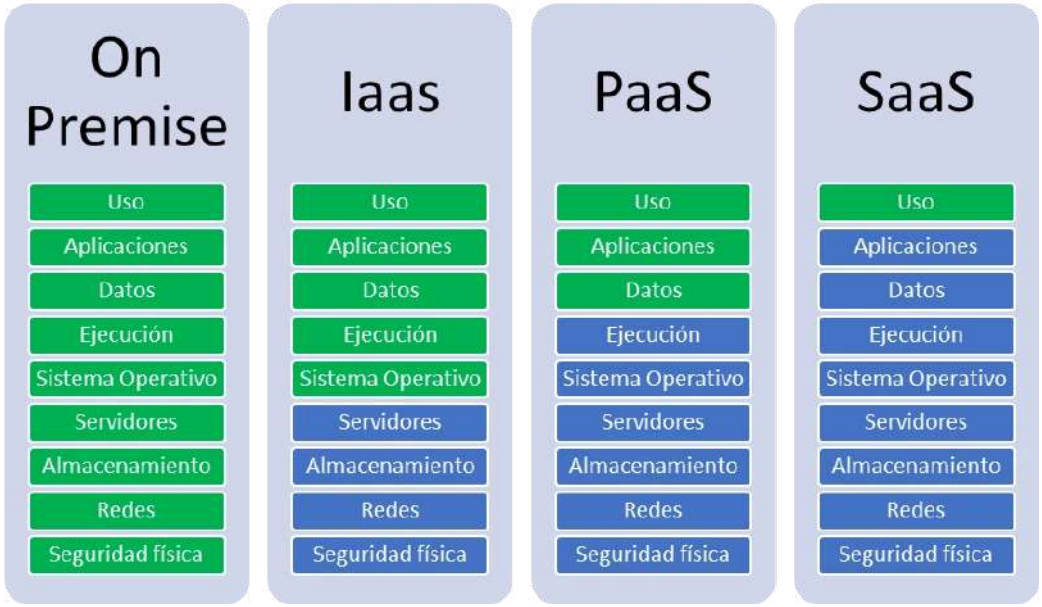
Ejemplo arquitectura con un orquestador swarm

ISP - El Proveedor de Servicios de Internet es la empresa que brinda conexión a Internet a sus clientes, conectando a sus usuarios a Internet a través de diferentes tecnologías como ADSL.

Cloud computing - Es la ejecución de las cargas de trabajo en las nubes, las cuales son entornos de TI que extraen, agrupan y comparten recursos flexibles en una red. El cloud computing y las nubes no son tecnologías en sí mismas, es una acción (la función que se encarga de ejecutar cierta carga de trabajo en una nube), son entornos (sitios donde se ejecutan aplicaciones) y son elementos (los sistemas de software y hardware que se utilizan para diseñar y usar las nubes).

VPS - El Servidor Privado Virtual es un servicio de alojamiento que utiliza la tecnología de virtualización para proporcionar recursos dedicados (privados) en un servidor con múltiples usuarios.

IaaS - Infraestructura como servicio. Se trata de un servicio de pago según el consumo, donde un tercero le presta los servicios de infraestructura, como el almacenamiento y la virtualización, cuando los necesita, a través de la nube o de Internet.



Opciones de cloud computing como servicio. **Casillas azules**: responsabilidad del proveedor cloud. **Casillas en verde**: responsabilidad del usuario o cliente.

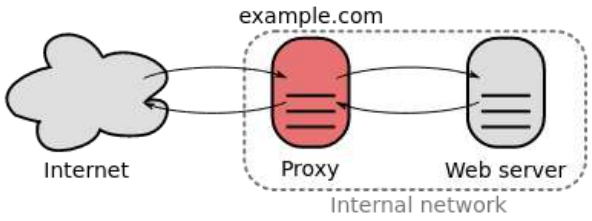
namespaces - Los espacios de nombres son una de las características del Kernel de Linux y un aspecto fundamental de los contenedores en Linux. Docker utiliza espacios de nombres de varios tipos para proporcionar el aislamiento que los contenedores necesitan para seguir siendo portátiles y no afectar al resto del sistema anfitrión. Cada aspecto de un contenedor se ejecuta en un espacio de nombres separado y su acceso está limitado a ese espacio de nombres.

cgroups - Proporcionan una capacidad de limitación de recursos y de información dentro del espacio del contenedor. Permiten un control granular sobre qué recursos del host se asignan a los contenedores y cuándo se asignan.

LXC - Es un conocido tiempo de ejecución de contenedores de Linux que consiste en herramientas, plantillas y enlaces de bibliotecas y lenguajes. Es de muy bajo nivel, muy flexible y cubre casi todas las características de contención soportadas por el kernel.

Balanceo de carga - En informática se refiere a la técnica usada para compartir el trabajo a realizar entre varios procesos, ordenadores, discos u otros recursos. Por ejemplo, es la manera en que las peticiones de Internet son distribuídas sobre una fila de servidores.

Proxy inverso - Oculta la complejidad de los servicios para los que hace de intermediario. Al situarse como intermediario es capaz de proporcionar funcionalidades adicionales como balanceo de carga, limitar el número de peticiones por unidad de tiempo, duplicar peticiones o realizar la autenticación entre otras funciones.



Proxy inverso

Circuit Breaker - Evita que una aplicación intente de manera reiterada una operación que con probabilidad vaya a fallar, permitiendo que esta continúe con su ejecución sin malgastar recursos mientras el problema no se resuelva. Además este patrón puede detectar cuando se ha resuelto el problema permitiendo de esta manera volver a ejecutar la operación comprometida.

Mirroring - La duplicación de datos para propósitos de backup o para distribuir el tráfico de la red entre varias computadoras (en este caso contenedores de aplicaciones) con los mismos datos.

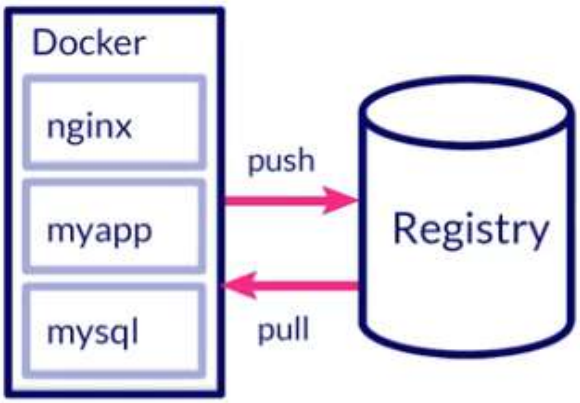
ACME - El entorno de gestión de certificados automatizado es un protocolo estándar para automatizar la validación, instalación y gestión de dominios de certificados X.509.
Estándar: https://www.rfc-editor.org/rfc/rfc5280.txt

Let's Encrypt - Es una autoridad de certificación que proporciona certificados X.509 gratuitos para el cifrado de Seguridad de nivel de transporte (TLS) a través de un proceso automatizado diseñado para eliminar el complejo proceso de creación manual, la validación, firma, instalación y renovación de los certificados de sitios web seguros.

OTP - One-Time Password. La “contraseña de un solo uso” es una contraseña que pierde su validez después de su uso, de ahí su denominación. Por lo general, se emplea como parte de una autenticación 2FA.

U2F - El segundo factor universal es un estándar abierto que refuerza y simplifica la autenticación de dos factores (2FA) mediante el uso de dispositivos USB especializados o de comunicación de campo cercano (NFC) basados en una tecnología de seguridad similar a la de las tarjetas inteligentes.

Ciclo de vida del desarrollo de software - Es la estructura que contiene los procesos, actividades y tareas relacionadas con el desarrollo y mantenimiento de un producto de software, abarcando la vida completa del sistema, desde la definición de los requisitos hasta la finalización de su uso.



Ciclo de vida ágil y eficiente de docker

Canalización CI/CD - Es un método para distribuir las aplicaciones a los clientes con frecuencia mediante el uso de la automatización en las etapas del desarrollo de aplicaciones. En concreto, el proceso de integración y distribución continuas incorpora la automatización y la supervisión permanentes en todo el ciclo de vida de las aplicaciones, desde las etapas de integración y prueba hasta las de distribución e implementación. Cuenta con el respaldo de los equipos de desarrollo y de operaciones que trabajan en conjunto de manera ágil, con un enfoque de DevOps o de ingeniería de confiabilidad del sitio (SRE).

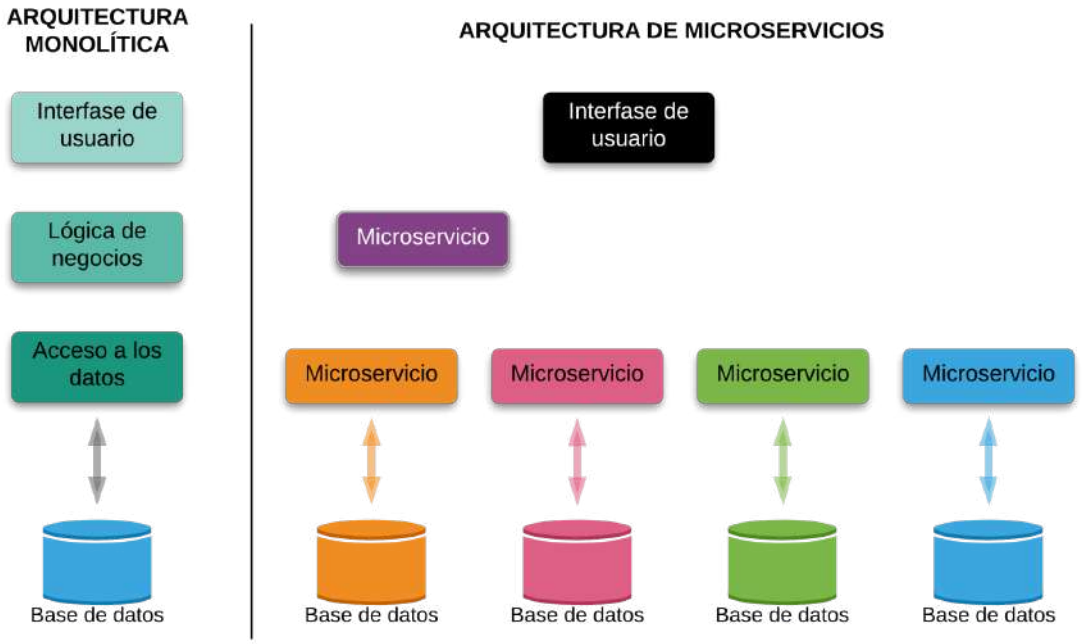
Proceso desde la creación hasta el despliegue en producción

Integración continua - Es una práctica habitual en desarrollo de software que consiste en integrar frecuentemente mejoras en el código de un proyecto una vez han sido validadas, normalmente varias veces al día, con el objetivo de detectar errores lo antes posible.

DevOps - Es una combinación de los términos ingleses development (desarrollo) y operations (operaciones). Permite que los roles que antes estaban aislados (desarrollo, operaciones de TI, ingeniería de la calidad y seguridad) se coordinen y colaboren para producir productos mejores y más confiables. Al adoptar una cultura de DevOps junto con prácticas y herramientas de DevOps, los equipos adquieren la capacidad de responder mejor a las necesidades de los clientes, aumentar la confianza en las aplicaciones que crean y alcanzar los objetivos empresariales en menos tiempo.

SRE - La ingeniería de confiabilidad del sitio es un enfoque de ingeniería de software para las operaciones de TI. Los equipos de SRE utilizan el software para gestionar los sistemas, resolver los problemas y automatizar las tareas operativas.

Sistema/aplicación monolito/monolítico - Los primeros programas informáticos utilizaban esta arquitectura, agrupando todo lo relacionado con el sistema dentro del mismo proyecto. Se caracteriza por que: los programas son fáciles de desarrollar, el despliegue y la ejecución del software son muy sencillos y, por último, el costo de desarrollo es bajo en comparación con otras arquitecturas. Los problemas de este tipo de arquitectura, como la escalabilidad o la dificultad para los desarrolladores (necesitan entender todo el código de la aplicación) han hecho que este tipo de desarrollo de software deje de ser utilizado en muchos proyectos (aunque su sencillez y bajo coste hace que siga siendo interesante para ciertos proyectos con bajos requerimientos en cuanto funcionalidades).



Comparativa de arquitectura monolítica con la de microservicios.

Drag & Drop - Significa arrastrar y soltar. Es una técnica que facilita la interacción del usuario con un programa o aplicación de manera intuitiva para llevar elementos de un lugar a otro. Muy utilizada por la facilidad que obtiene el usuario inexperto y por la proliferación de las pantallas táctiles.

Shell - Es una interfaz de línea de comandos, lo que significa que está basado en texto. El usuario puede escribir comandos u órdenes para realizar funciones como ejecutar programas, abrir y navegar directorios y/o ver los procesos que se están ejecutando a tiempo real. Dado que el shell está solo una capa por encima del sistema operativo, puede realizar operaciones que no siempre son posibles utilizando la interfaz gráfica (GUI).

GUI - Es un programa informático que actúa de interfaz de usuario, utilizando un conjunto de imágenes y objetos gráficos para representar la información y acciones disponibles en la interfaz. Su principal uso consiste en proporcionar un entorno visual sencillo para permitir la comunicación con el SO de una computadora.

Obsolescencia programada - Es la determinación o programación del fin de la vida útil de un producto, de modo que, tras un período de tiempo calculado de antemano por el fabricante o por la empresa durante la fase de diseño del mismo, éste se torne obsoleto, no

funcional, inútil o inservible por diversos procedimientos, por ejemplo por falta de repuestos, induciendo al usuario a la compra de un nuevo producto que lo sustituya.

Explicación gráfica de la obsolescencia programada

Pruebas unitarias - Es una forma de comprobar el correcto funcionamiento de una unidad de código. Por ejemplo, en el diseño estructurado o en diseño funcional una función o un procedimiento. Esto sirve para asegurar que cada unidad funcione correctamente y eficientemente por separado. Además de verificar que el código hace lo que tiene que hacer, verificamos que sea correcto el nombre, los nombres y tipos de los parámetros, el tipo de lo que se devuelve, que si el estado inicial es válido, entonces el estado final es válido también.

Servidor perimetral de CDN - es un ordenador que existe en el extremo lógico o "perímetro" de una red. Suele servir de conexión entre redes distintas. Uno de los objetivos principales de un servidor perimetral de CDN es almacenar el contenido lo más cerca posible de la máquina del cliente que realiza la solicitud, reduciendo así la latencia y mejorando los tiempos de carga de la página.

IA - Es la inteligencia expresada por máquinas, sus procesadores y sus softwares, que serían los análogos al cuerpo, el cerebro y la mente, respectivamente. En ciencias de la computación, una máquina «inteligente» ideal es un agente flexible que percibe su entorno y lleva a cabo acciones que maximizan sus posibilidades de éxito en algún objetivo o tarea.

Machine Learning - El aprendizaje automático o aprendizaje automatizado es el subcampo de las ciencias de la computación y una rama de la inteligencia artificial, cuyo objetivo es desarrollar técnicas que permitan que las computadoras aprendan. En muchas ocasiones el campo de actuación del aprendizaje automático se solapa con el de la estadística inferencial, ya que las dos disciplinas se basan en el análisis de datos. Sin embargo, el aprendizaje automático incorpora las preocupaciones de la complejidad computacional de los problemas.

4. Objetivos

Como se ha comentado en el planteamiento inicial, el desarrollo de este proyecto está separado en 2 partes que se describirán a continuación. Los objetivos que se concretarán en la primera parte se han evaluado en el entorno de pruebas, comprobando su viabilidad y después de todo el proceso de investigación se han considerado adecuados técnica y funcionalmente.

4.1. Primera parte: Servidor en producción, proveedor de servicios Open Source

Los objetivos se pueden esquematizar en los siguientes pasos:

- Contratación y configuración del dominio y el servicio hosting
- Configuración y preparación del servidor en producción. Incluyendo:
    - Actualizaciones
    - Instalación de software
    - Gestión de la red
    - Creación de usuarios
    - Securización de ssh
- Instalación de Docker y despliegue de contenedores base: Portainer y Traefik.
- Automatización de los certificados SSL
- Despliegue de aplicaciones
    - Stack servicio VPN.
    - Stack de servicio de documentación.
    - Stack de Wordpress.
    - Stack de mailserver.
    - Stack de herramienta de métricas.
    - Stack de servicio de nube.
    - Stack de servicio de chats.
    - Stack de servicio de llamadas.
    - Stack de servicio de videollamadas.
- Configuración de la conexión perimetral.

Objetivos post-proyecto:

- Desarrollo, configuración y maquetación de cada uno de las aplicaciones.
- Testeo de funcionalidad, recogida y evaluación de los resultados.

4.2. Posible segunda parte: Proveer a un smartphone con los servicios del servidor

Esta segunda parte se basa en **rootear un smartphone e instalar los servicios clientes de código abierto**. En el presente proyecto no se incluye pero en un futuro a medio plazo estará disponible en https://gitea.vergaracarmona.es/manudocker. Sin entrar en mucho detalle, a modo especulativo, he redactado algunas conclusiones como “notas para el futuro”.

4.2.1. ¿Por qué rootear un móvil?

Las **razones** por las que rootear Android son:

- Aunque el núcleo de este SO sigue siendo parte del proyecto de código abierto, la mayoría de las **aplicaciones principales son de código privativo**.
- Cada vez hay más **bibliotecas y APIs** que sólo están disponibles en teléfonos con preinstalación de apps Google, lo que **limita las aplicaciones de terceros al ecosistema de Google**.
- La mayoría de empresas dedicadas al desarrollo del SW y del HW1 de Smartphone son Big Tech. De cierta manera, se han adueñado del código abierto de este SO, creando **versiones Android con un mayor porcentaje de código privativo**.

Por estas razones, Android se describe como un **sistema abierto del tipo "míra pero no toques"** (Descripción extendida en comunidades de software libre).

4.2.2. Sistemas Operativos Libres para Smartphones

Los posibles SO libres para smartphones que pueden sustituir a Android y que me han parecido más idóneos son los siguientes:

- **Lineage OS** https://www.lineageos.org/
- **Graphene OS** https://grapheneos.org/
- **Calyx OS** https://calyxos.org/

Otra línea de investigación son las herramientas tipo **AOSP** (Es lo más llano para construir sobre ello) o **ResurrectionRemic.**

Para evitar que aplicaciones populares de código abierto utilicen bibliotecas propietarias de Google se puede implementar **MicroG Project** https://microg.org/. Es un **clon de software libre de las bibliotecas y aplicaciones centrales** propietarias de Google. Puede servir de complemento para los SO libres mencionados anteriormente. La mayoría de los componentes de microG están lejos de ser completos, pero los usuarios están sorprendidos por los resultados.

4.2.3. Aplicaciones libres para smartphones

Para evitar utilizar las Apps de Google, se puede utilizar la app **f-droid** para descargar desde allí el resto de apps. Es un **repositorio de software** para dispositivos Android que funciona de manera similar a la tienda de Google Play, pero solo contiene software libre y de código abierto. Las apps pueden buscarse e instalarse desde la web de F-Droid o desde el cliente oficial (el cual no está disponible en Google Play, pues este no admite tiendas alternativas, pero puede ser instalado directamente mediante su archivo APK).

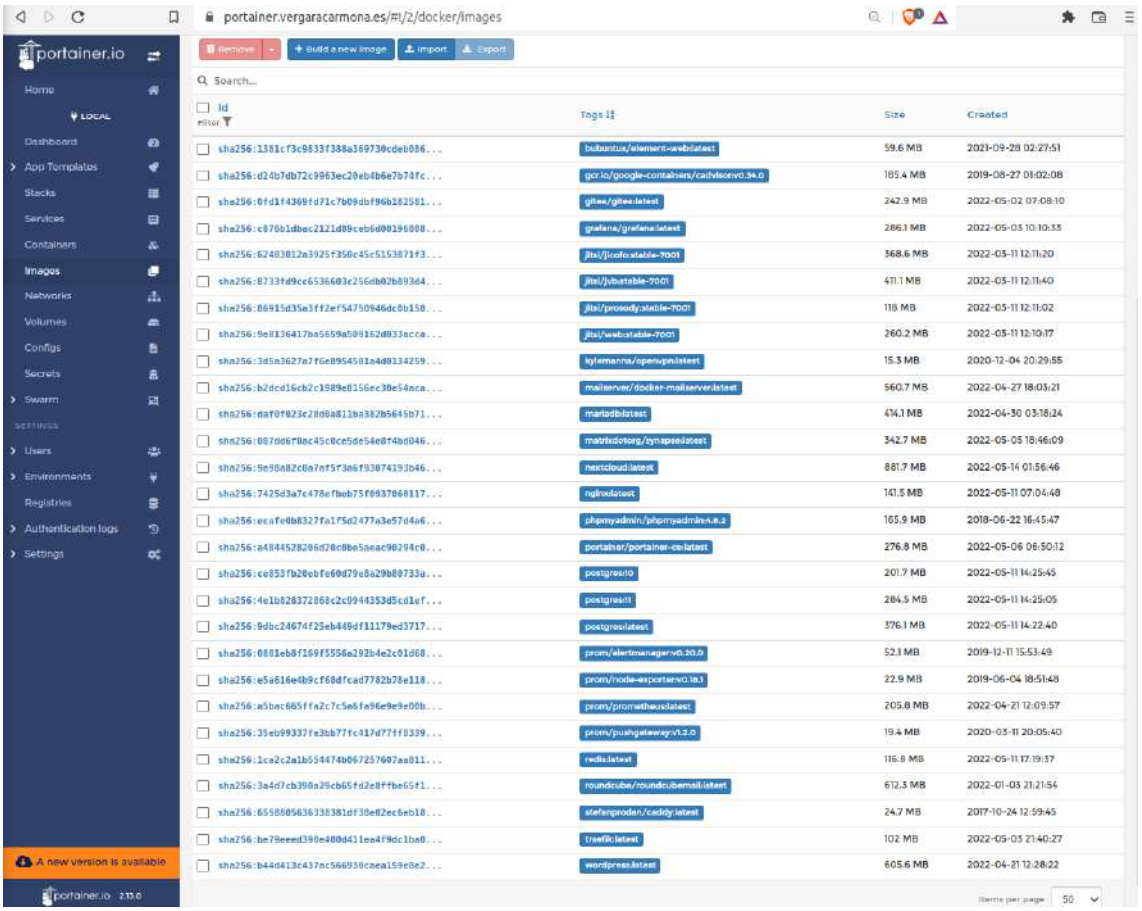
5. Despliegue del proyecto

Llegado a este punto y para alcanzar los objetivos, se deben **desplegar las imágenes y contenedores** considerados como idóneos, del servidor de desarrollo al de producción.

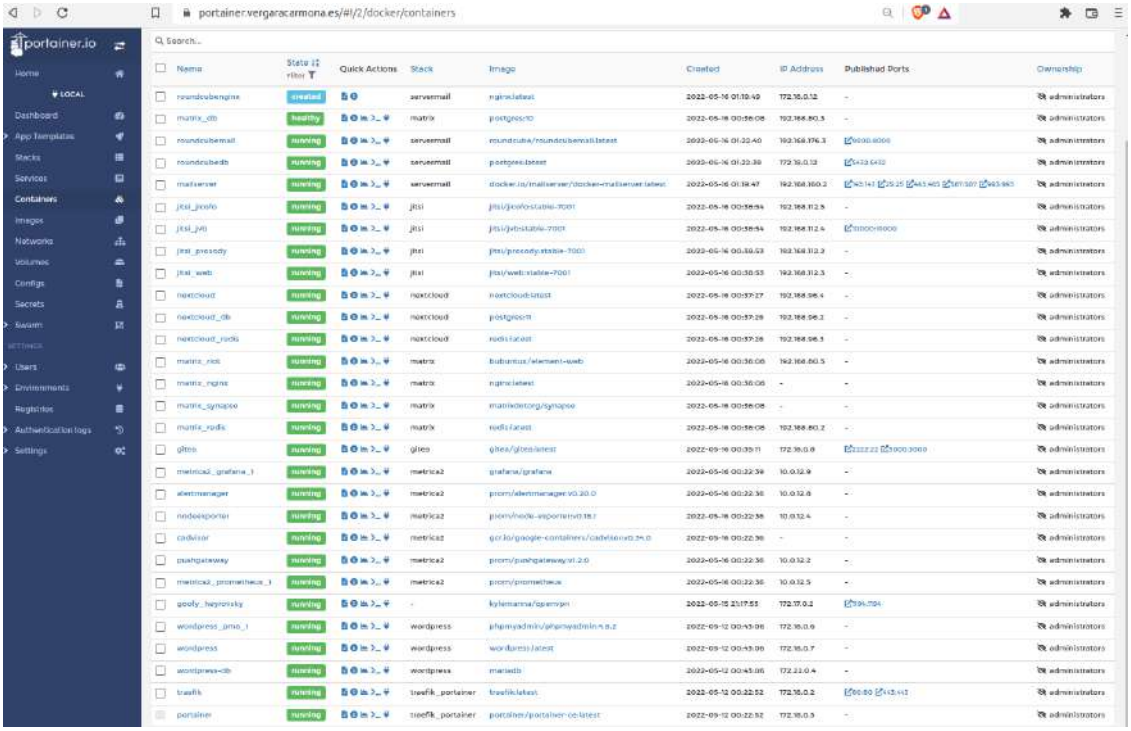
Los servicios están contenidos en imágenes/contenedores de aplicación docker que se despliegan mediante docker-compose. Se orquestan con stacks de swarm y se podrán monitorizar con la interfaz gráfica de la webapp Portainer.

Las capturas más relevantes del servidor de desarrollo una vez desplegado son las siguientes:

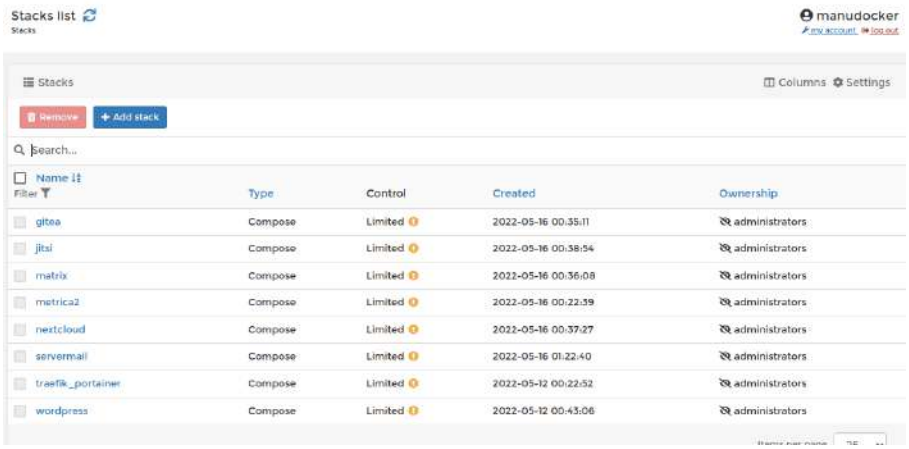
- Listado en el servidor de pruebas de las imágenes docker que utilizan los contenedores.



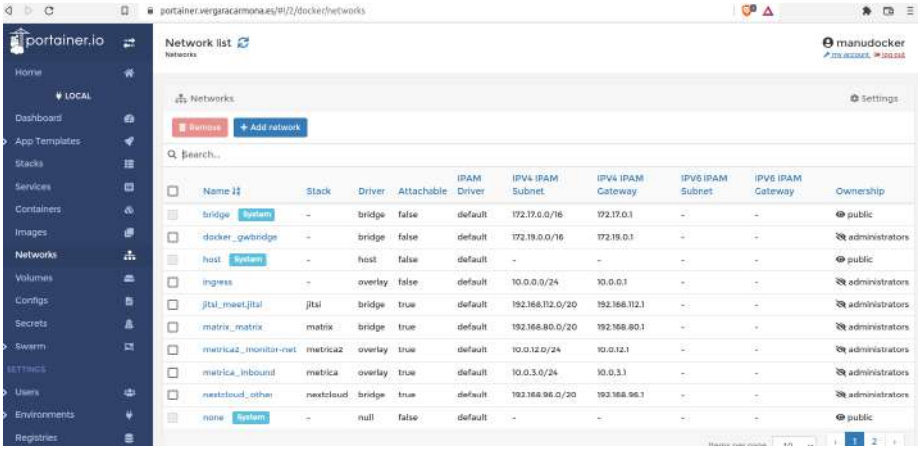
- Listado de los contenedores desplegados.



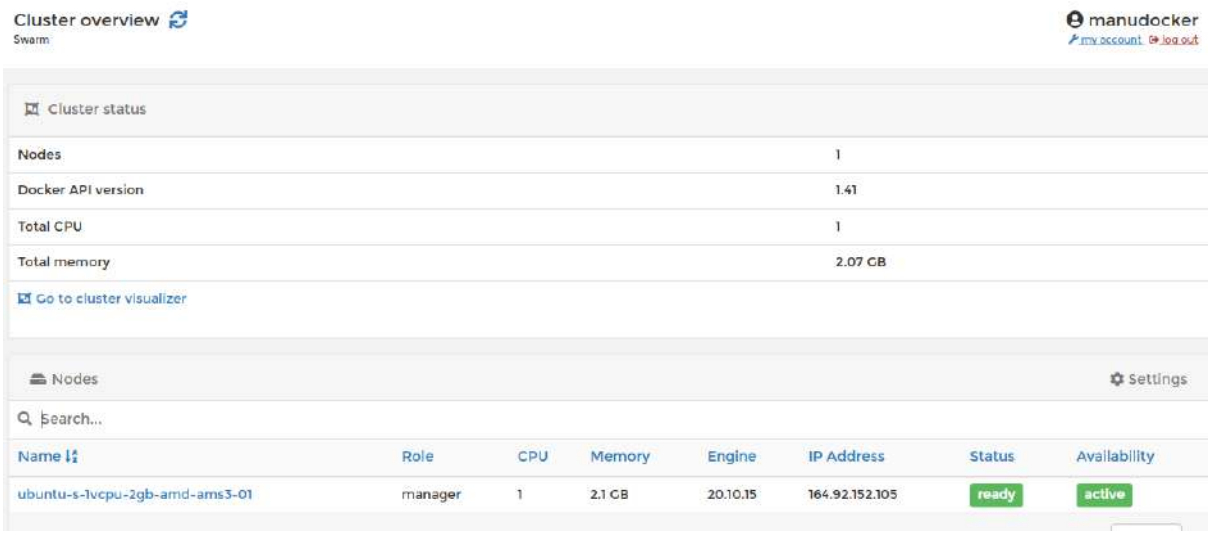
- Listado de stack desde Portainer



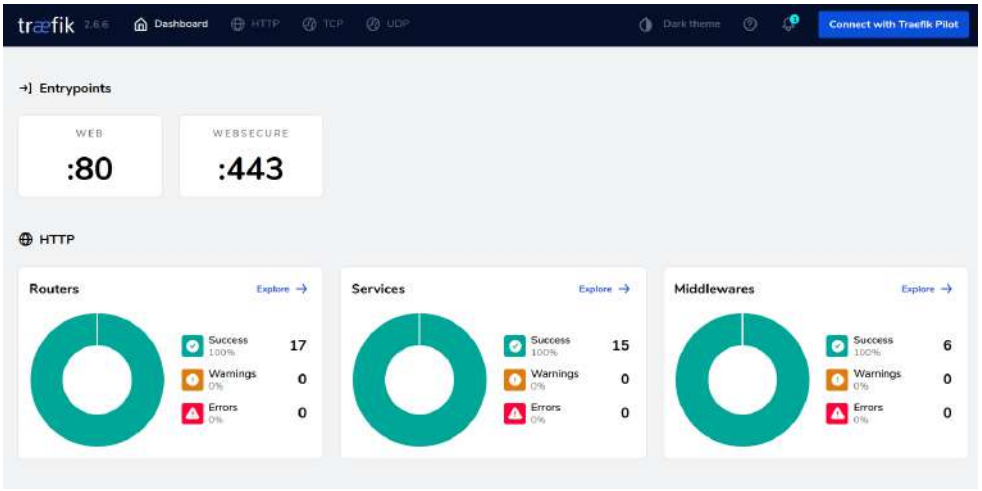
- Listados de las redes empleadas en docker.

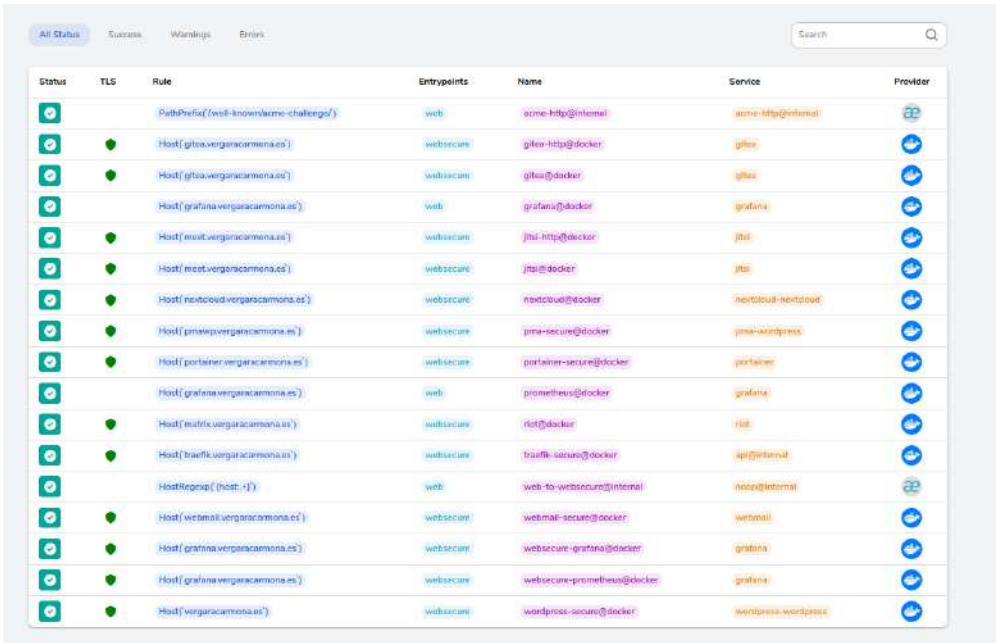


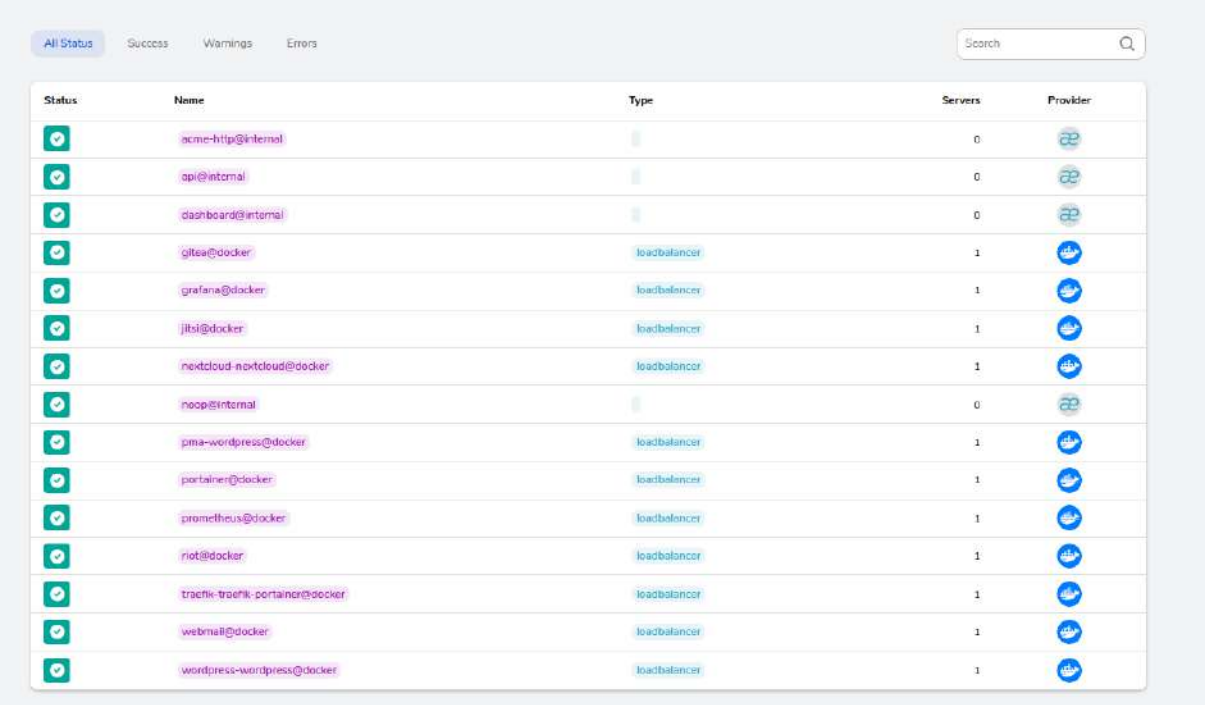
- Listado de los nodos del orquestador swarm.

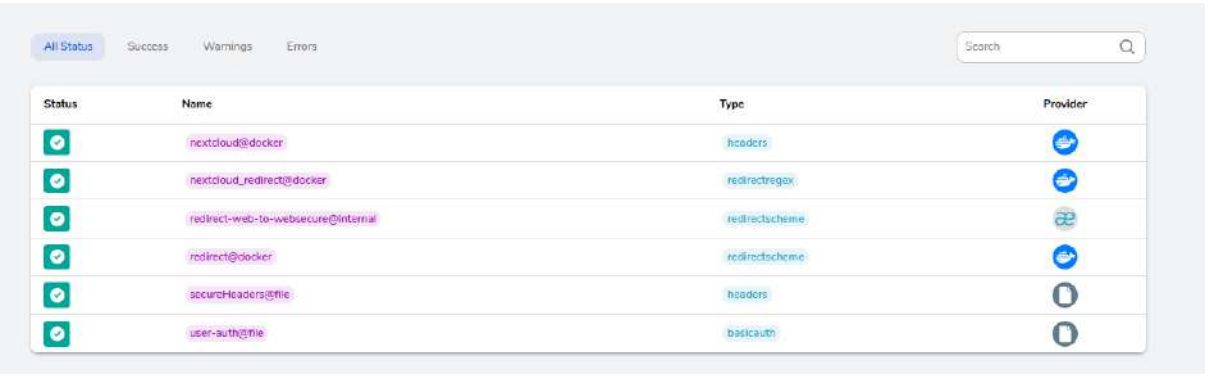


- Redes en el proxy inverso Traefik

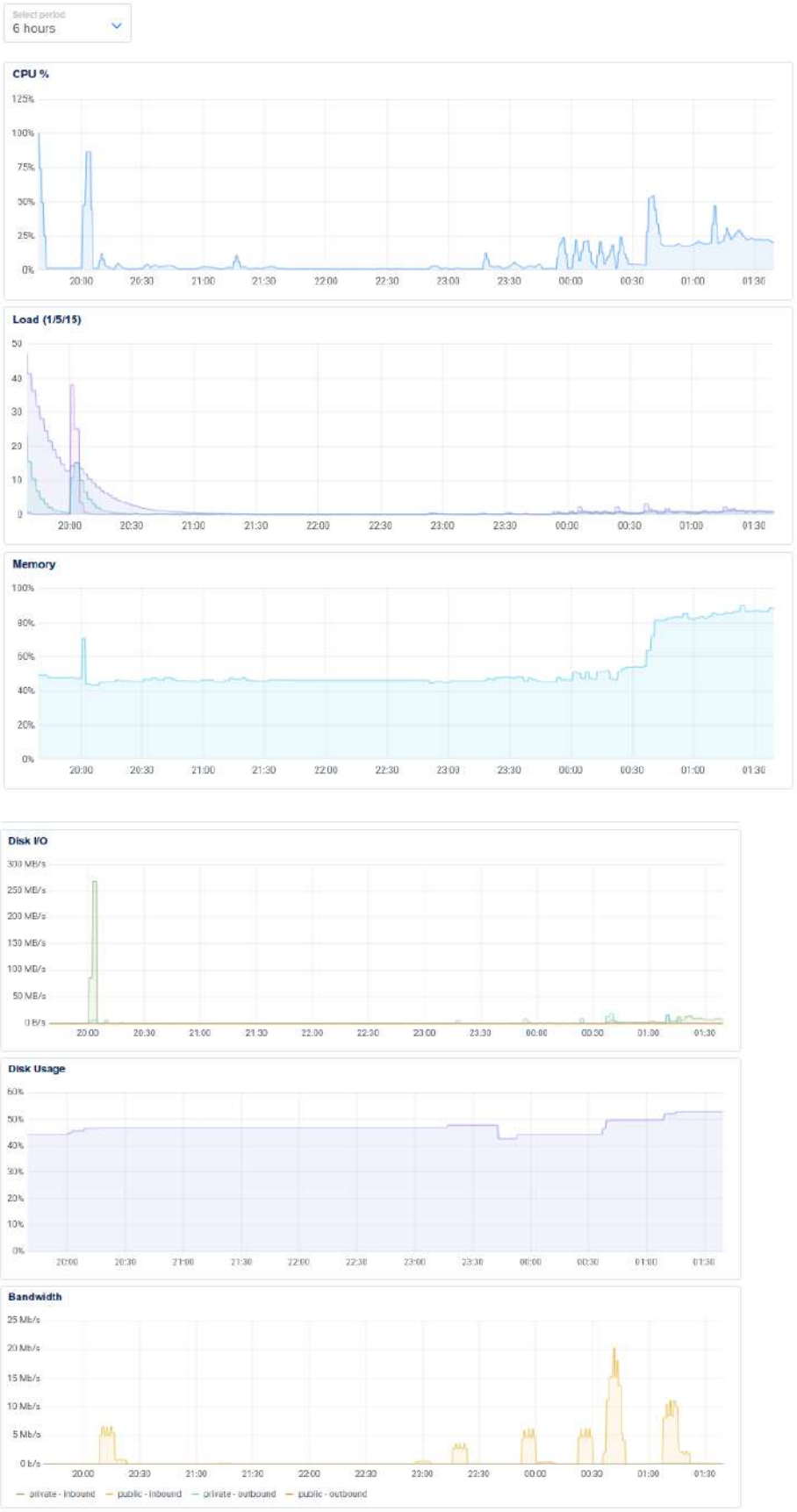





All Status
Success
Warnings
Errors
Search
Status
Name
Type
Servers
Provider
acme-http@internal
api@internal
dashboard@internal
gitea@docker
grafana@docker
jitsi@docker
nextcloud-nextcloud@docker
noop@internal
pma-wordpress@docker
portainer@docker
prometheus@docker
riot@docker
traefik-traefik-portainer@docker
webmail@docker
wordpress-wordpress@docker
loadbalancer


All Status
Success
Warnings
Errors
Search
Status
Name
Type
Provider
nextcloud@docker
nextcloud_redirect@docker
redirect-web-to-websecure@internal
redirect@docker
secureHeaders@file
user-auth@file
headers
redirectregex
redirectscheme
basicauth

- Monitor CPU, carga, Entrada/Salida, uso de disco y ancho de banda de DigitalOcean



5.1. Selección y configuración del servidor VPS

He seleccionado **3 de las empresas** que ofrecen servidores IaaS que me han parecido más relevantes y he hecho un resumen de sus **VPS** que se adaptan al proyecto.

	vultr https://www.vultr.com/			**DigitalOcean** https://digitalocean.com			**Clouding** https://clouding.io/		
vCPUs	1 AMD	2 AMD	4 AMD	**1 AMD**	2 AMD	4 AMD	1 Intel	2 Intel	4 Intel
RAM	2 GB	4 GB	8 GB	**2 GB**	4 GB	8 GB	2 GB	4 GB	8 GB
SSD	50 GB	100 GB	180 GB	**50 GB**	80 GB	160 GB	50 GB	80 GB	160 GB
Ancho de banda	3 TB	5 TB	6 TB	**2 TB**	4 TB	5 TB	4 TB	4 TB	4 TB
Precio mes	$ 12 (10,8 €)	$ 24 (21,6 €)	$ 48 (43,2 €)	**$ 12 (10,8 €)**	$ 24 (21,6 €)	$ 48 (43,2 €)	10 €	18 €	36,50 €
Precio hora	$ 0,018 (0,016 €)	$ 0,036 (0,032 €)	$ 0,071 (0,064 €)	**$ 0,018 (0,016 €)**	$ 0,036 (0,032 €)	$ 0,071 (0,064 €)	0,013 €	0,025 €	0,05 €

Se ha escogido el servidor VPS de **DigitalOcean**, de 1 cpu AMD con 2 GB de ram y con 50GB de almacenamiento. A pesar que el resto ofrecen más ancho de banda, es el **más estable** según he podido comprobar en foros y con colegas. La diferencia de precio no es significativa. Tiene **facilidad para escalar** el servicio y la **integración con aplicaciones**, además de **monitorización** propia de los componentes del servidor.

Tutorial de creación de un servidor en Digital Ocean:
https://vergaracarmona.es/crear-un-vps-droplet-en-digital-ocean/

Después de adquirir el servicio he conocido la existencia de otros que podrían ser candidatos: **Kimsufi** (outlet de OVH) y **netcup**. No se han estudiado a fondo.

Para el dominio ya tengo adquirido vergaracarmona.es en el proveedor de servicios de dominios y hosting **Raiola Networks,** a quienes les tenga una especial confianza por llevar mucho tiempo trabajando con ellos. Con cada nuevo servicio se explicarán las DNS necesarias, pero para configurar las básicas de un servidor y añadir el dominio en DigitalOcean se puede ver el Tutorial “Configuración de las DNS de un dominio de Raiola en Digital Ocean”:
https://vergaracarmona.es/configuracion-de-las-dns-de-un-dominio-de-raiola-en-digital-ocean

5.2. Actualización/Configuración del server. Instalaciones base

Una vez creado el **servidor VPS** los pasos a seguir son semejantes a los ya seguidos en la instalación del laboratorio de pruebas en la MV, pero esta vez emplearemos el **SO ubuntu 22.04 LTS**. Los pasos que se siguen del tutorial "Preparación de entorno de desarrollo local" (https://vergaracarmona.es/preparacion-de-entorno-de-pruebas-local-para-docker/) son los siguientes:

1. Revisar que los **repositorios** son los correctos en `sources.list`. Incluyen mirrors de DigitalOcean: http://mirrors.digitalocean.com/ubuntu/
2. Comprobar las **actualizaciones** y aplicarlas.
3. Instalar los paquetes necesarios para la **configuración del idioma**.
4. **Reinicio del SO** para asegurar que se apliquen los cambios. (Una vez se lance a producción no se deberá reiniciar más salvo extrema necesidad).
5. Instalación de los **paquetes básicos** unzip, zip, tree, curl, ufw
6. Configuración **cortafuegos ufw** (En un inicio se permitirán los protocolos ssh, http, https, smtp e imap. En el caso que fuere necesario otros servicios se habilitarán más adelante.)

Para no usar el usuario root, por motivos de seguridad, **se crea un usuario principal** administrador con los privilegios sudo. (Tutorial "Crear un nuevo usuario habilitado para sudo en Ubuntu"
https://vergaracarmona.es/crear-un-nuevo-usuario-habilitado-para-sudo-en-ubuntu/)

Acto seguido, se configura el **servicio ssh** para que sea seguro en el documento `/etc/ssh/sshd_config` con los siguientes parámetros:

- `Port 432` (o el que se quiera menor a 1024)
- `Protocol 2`
- `LoginGraceTime 30`
- `PermitRootLogin no`
- `MaxAuthTries 2`
- `MaxStartups 3`
- `AllowUsers <usuario>`
- `PermitEmptyPasswords no`

También configuro un **banner**. Para aplicar los cambios se ejecuta el siguiente comando:

```
/etc/init.d/ssh restart
```

Se debe asegurar que exista el doc `~/.ssh/authorized_keys` en la carpeta home del usuario que se quiera autorizar. En este documento deben constar las claves públicas autorizadas.

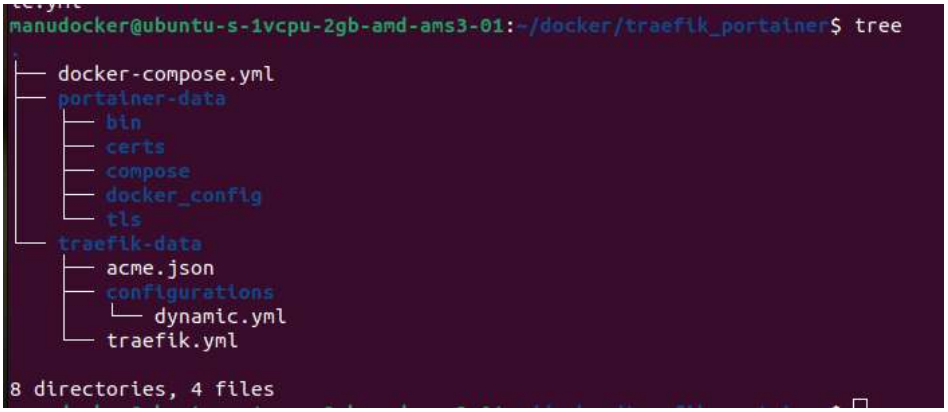
Ahora, se realiza la **instalación de docker**, siguiendo el paso 4.2 del Tutorial "Preparación de entorno de desarrollo local para pruebas con docker":
https://vergaracarmona.es/preparacion-de-entorno-de-pruebas-local-para-docker/

Y por último, se instala **docker-compose** para facilitar la utilización de este plugin:

```
sudo apt install docker-compose
```

5.3. Despliegue del proxy inverso y del gestor de orquestación.

Para un despliegue más ágil, he creado en la fase de desarrollo un **docker-compose** que incluye el despliegue del **proxy inverso Traefik** y del **gestor de orquestación portainer.** La estructura para el despliegue es la siguiente:



En esta misma estructura podría incluir el despliegue de todas las aplicaciones dentro del directorio `traefik_portainer`, incluso desplegarlas desde el mismo `docker-compose.yml` utilizado como base, pero he preferido separarlas en **stacks**, cada uno con su fichero `yaml` y su estructura de archivos. De este modo se facilita la gestión por separado de cada webapp (desplegar, bajar, actualizar, modificar, etc).

Los pasos a seguir para el despliegue del proxy y del gestor son:

1. Configuración de los **registros DNS**.

2. Descarga de la **estructura de archivos** y carpetas mediante `git clone`.
3. Creación de las **credenciales**

```
sudo apt install apache2-utils    # He tenido que instalar apache2
echo $(htpasswd -nb <nombre de usuario> <contraseña>)
```

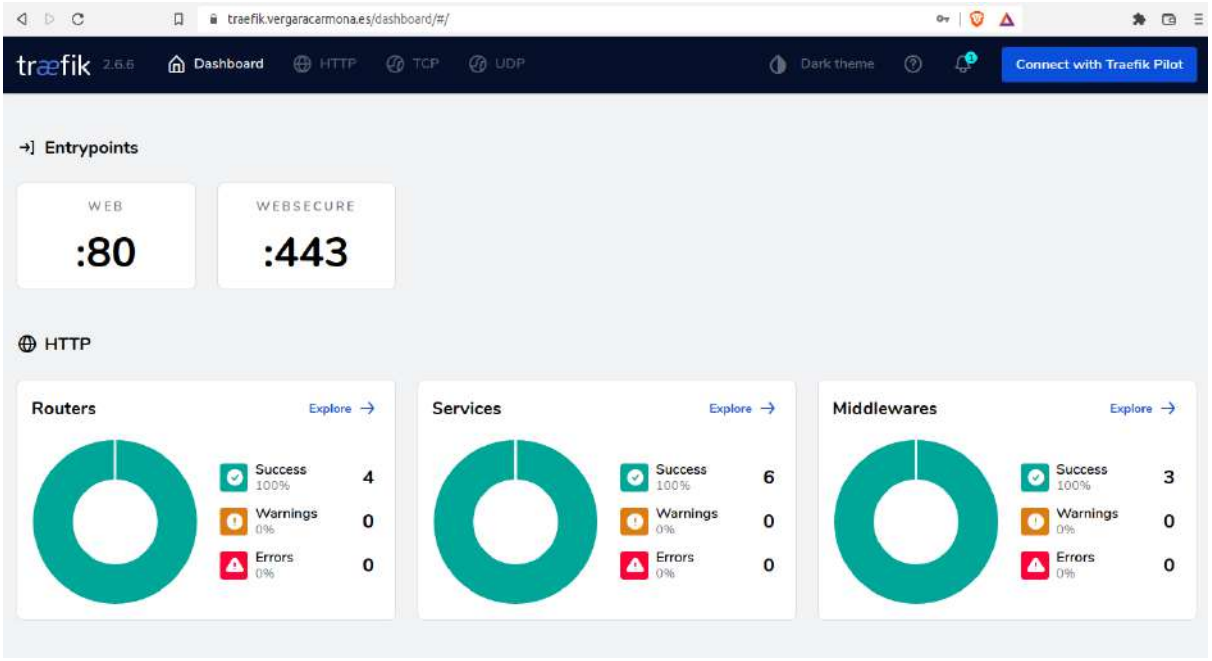
   Copiar el resultado que arroja la terminal y pegar en el documento `dynamic.yml`
4. Creación de la **red proxy** de traefik, que será la conexión al exterior de todos los contenedores frontend.
5. Adaptación del documento `docker-compose.yml` al servidor.
6. Dar los **permisos adecuados** a acme.json
7. Ejecutar la pila con: `docker-composer up -d`
8. **Iniciamos las webapps** entrando en la url.

Para más detalles de cada uno de los pasos de despliegue y de la estructura de fichero y archivos, se puede consultar el tutorial: "*Desplegar con docker-compose los servicios Traefik y Portainer del desarrollo de la práctica*": https://vergaracarmona.es/desplegar-con-docker-compose-los-servicios-traefik-y-portainer/

Enlace de archivos en git: ENLACE GIT ####################

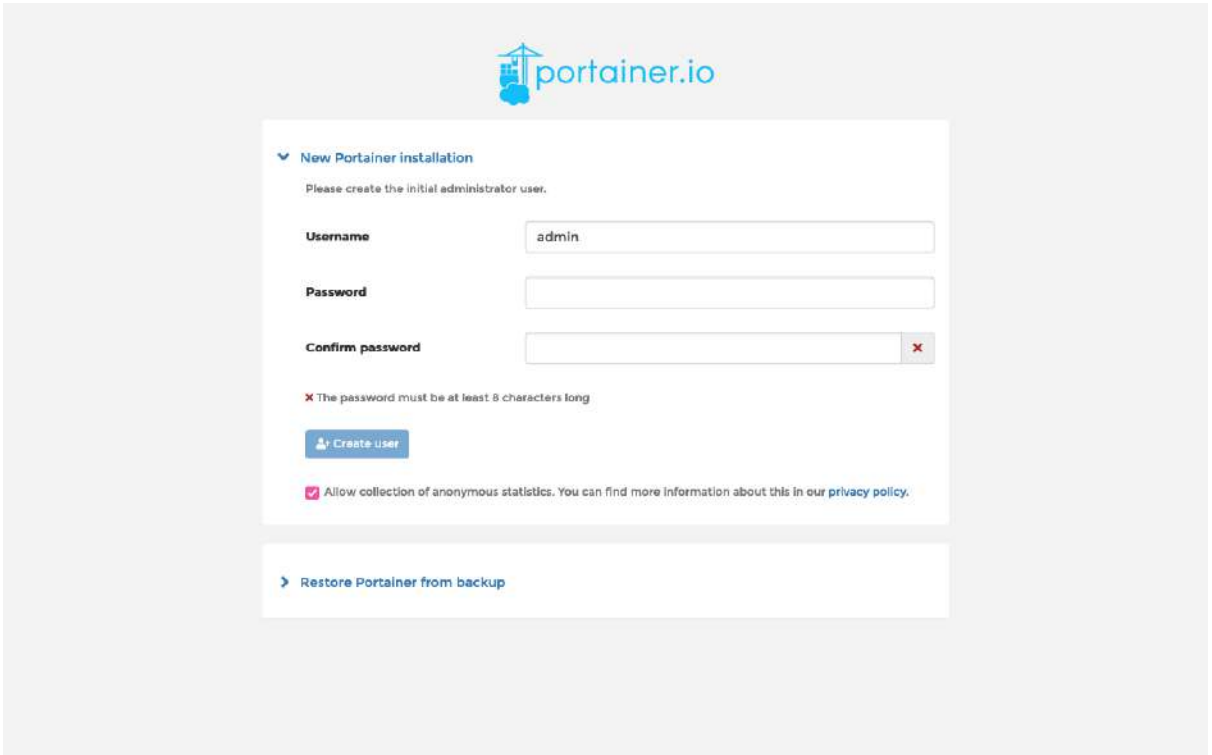
5.3.1. Traefik

Al entrar en la url https://traefik.vergaracamona.es nos pedirá las credenciales configuradas en el fichero `dynamic.yml` y entraremos en el panel de control:

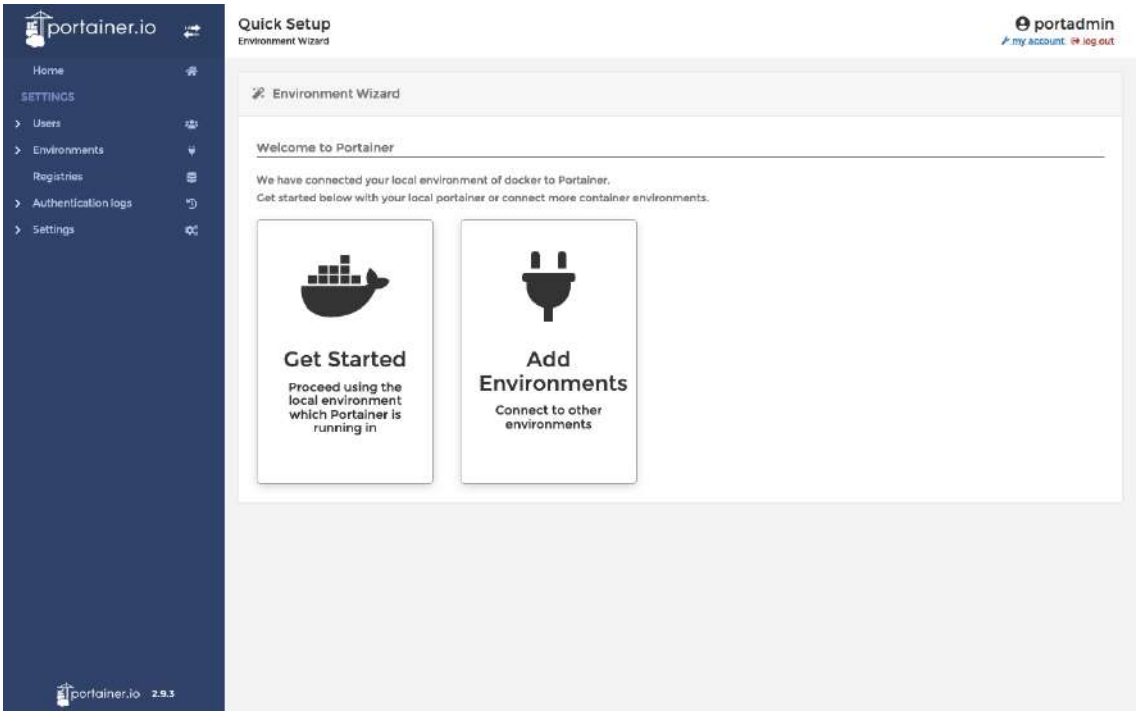


5.3.2. Portainer

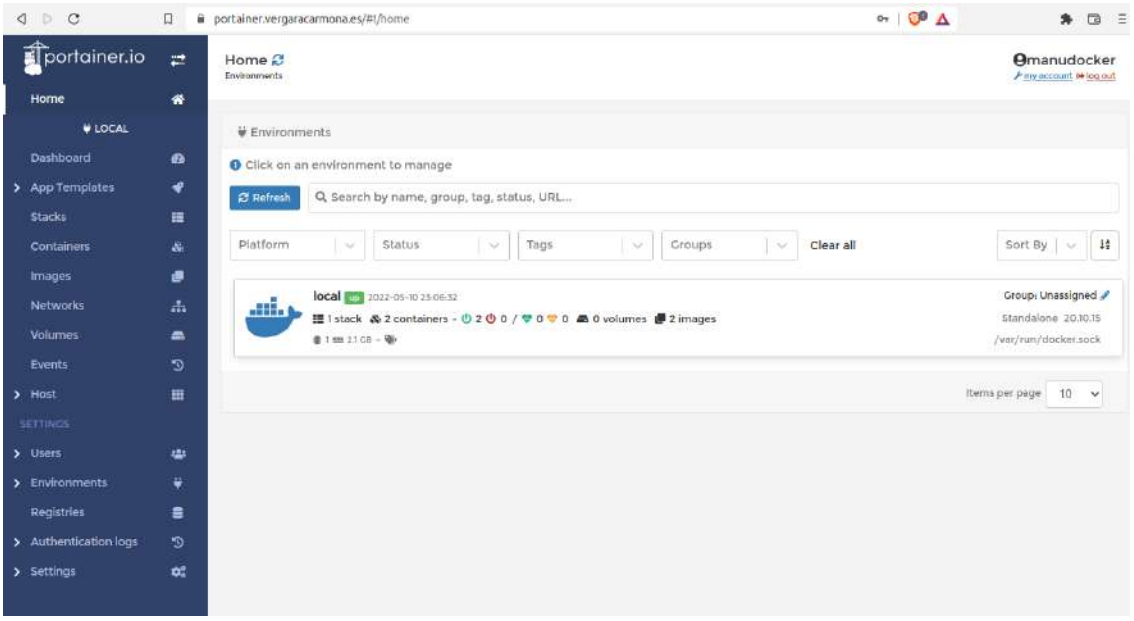
Al entrar en la url https://portainer.vergaracarmona.es nos solicita crear un usuario y una contraseña segura para este.



Por motivos evidentes de seguridad, conviene no crear el usuario “admin”. Después aparecerá un asistente rápido de configuración donde se clica “get docker”.



Y ya se podrá entrar en el panel de control.



5.3.3. Certificados SSL

Traefik soporta de forma **nativa** gestión de **certificados de proveedores ACME** como **Let’s Encrypt** así como certificados **actualizables de forma dinámica** definidos por el usuario. Además, se puede crear la **redirección automática de http a https** con una única configuración para todos los subdominios que se introduzcan. Todo esto es una **seguridad añadida** a la capa de contenedores configurados como frontend, porque nunca tendrán conexión directa con la red exterior.

5.4. Despliegue de aplicaciones

Los despliegue de cada una de las webapps se han conseguido a través de los siguientes pasos comunes:

1. Preparación en entorno de pruebas.
    a. Creación del Docker-compose con la configuración necesaria según la aplicación.
    b. Creación de estructura de directorios.
    c. Creación y configuración de documentos auxiliares como variables de entornos, DockerFile, registros, etc
    d. Configuración y maquetación de la aplicación así, si es necesario.
    e. Pruebas unitarias
2. Correr la aplicación en el servidor de producción. En algunos casos requiere de adaptación de los documentos, resolución de incidencias y/o configuraciones extras no contempladas. Esto último es básicamente porque el entorno de desarrollo creado no cumple con el principio básico de que debe estar en un entorno idéntico al de producción. No es lo mismo una MV en local que un VPS, aunque ambas sean MV tienen una serie de configuraciones que las diferencia.

```
version: "3"
services:
  web:
    build: web
    command: python app.py
    ports:
     - "5000:5000"
    volumes:
     - ./web:/code # modified here to take into account the new app path
    links:
     - redis
    environment:
     - DATADOG_HOST=datadog # used by the web app to initialize the Datadog library
  redis:
    image: redis
  # agent section
  datadog:
    build: datadog
    links:
     - redis # ensures that redis is a host that the container can find
     - web # ensures that the web app can send metrics
    environment:
     - DD_API_KEY=__your_datadog_api_key_here__
    volumes:
     - /var/run/docker.sock:/var/run/docker.sock
     - /proc/:/host/proc/:ro
     - /sys/fs/cgroup:/host/sys/fs/cgroup:ro
```

Ejemplo de un docker-compose básico con tres servicios: Una app python, redis y datadog.

```
FROM microsoft/dotnet:sdk AS build-env
WORKDIR /Docker

# Copy csproj and restore as distinct layers

RUN dotnet restore

# Copy everything else and build
COPY . ./
RUN dotnet publish -c Release -o out


# Build runtime image
FROM microsoft/dotnet:aspnetcore-runtime
WORKDIR /app
COPY --from=build-env /app/out .
ENTRYPOINT ["dotnet", "aspnetapp.dll"]
```

Ejemplo de un DockerFile con la librería SDK y .NET

Existen otros pasos comunes para finalizar la completa implementación de las webapps:

- Terminar los últimos detalles de la configuración y maquetación de cada aplicación.
- Darles funcionalidades extra a través de extensiones, plugins o similares, con la configuración extra que supone.
- Testeo de funcionalidades.
- Revisión de enlaces, buscar roturas y fallas de los certificados SSL.
- Comprobaciones de seguridad.
- Pruebas con usuarios reales.
- Recogida de resultados, conclusiones y posible reconfiguración.

Estos puntos no se han detallado dentro del presente proyecto por no ser relevantes para cumplir el objetivo general, que tan solo engloba la demostración de que es posible sustituir los servicios que reciben los smartphones de los Big Tech. De todos modos, se llevarán a cabo a posteriori en la mayoría de servicios, ya que serán utilizados de manera personal.

Tan solo se han añadido tutoriales con más detalles a los servicios con complicaciones técnicas.

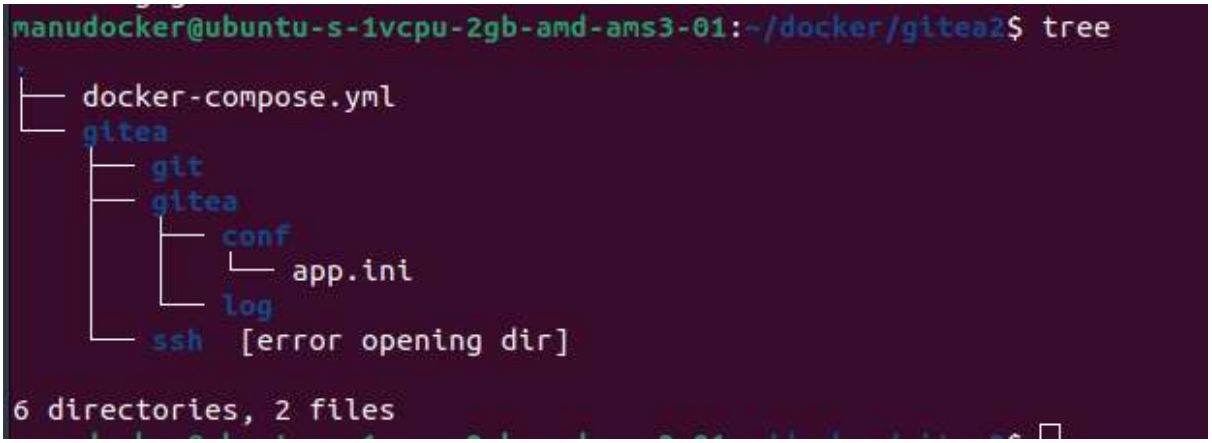
5.4.1. Servidor de documentación

Las tecnologías de contenedores hacen muy ágil la **portabilidad de las webapps** sin apenas configuración adicional. Los directorios y ficheros se pueden guardar en un servidor de documentación para así poder desplegar la estructura personalizada con un simple comando. Por este motivo, el primer servicio que instalaremos será **gitea**.

Los pasos a seguir para el despliegue del gitea son:

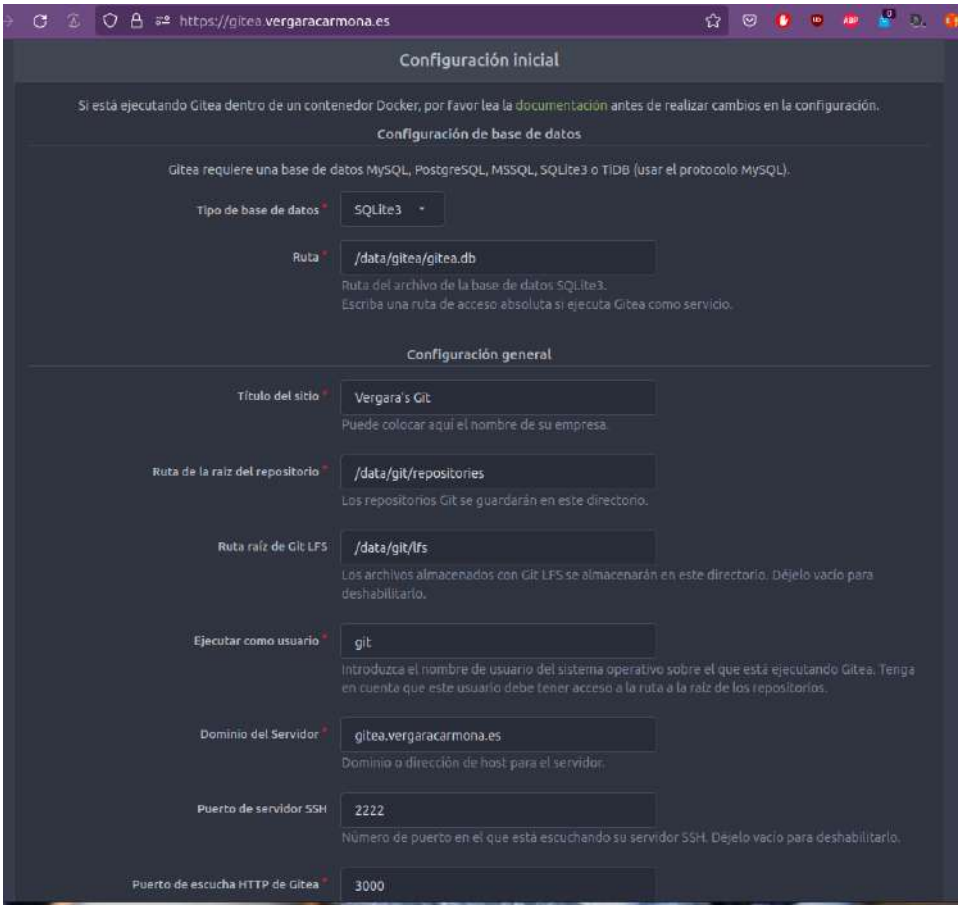
1. Configuración del **DNS**

2. Descarga de la **estructura de archivos** y carpetas mediante `git clone`.



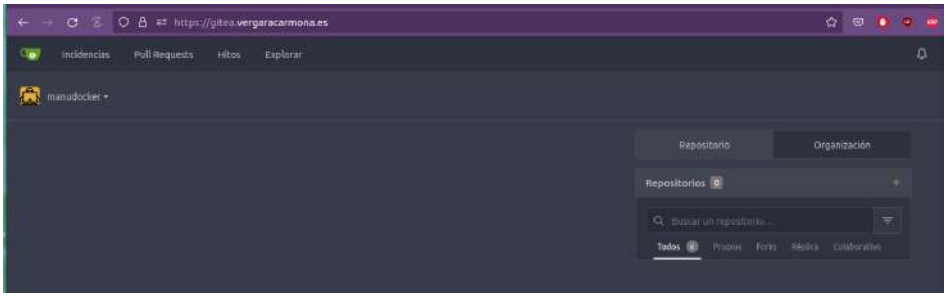
3. Adaptación del documento `docker-compose.yml` al servidor.
4. Dar los permisos adecuados a la estructura de directorios y ficheros públicos en `gitea`.
5. Ejecutar la pila con: `docker-composer up -d`

Al entrar en https://gitea.vergaracarmona.es nos pide la **configuración** de algunas características del servicio:



Es importante configurar la **cuenta de administrador**. Después clicamos en Instalar

Entonces ya podemos entrar en el panel personal:



5.4.2. Servicio VPN

OpenVPN es una herramienta de conectividad basada en software libre: **SSL, VPN Virtual Private Network**. OpenVPN ofrece conectividad **punto-a-punto** con validación jerárquica de usuarios y host conectados remotamente.

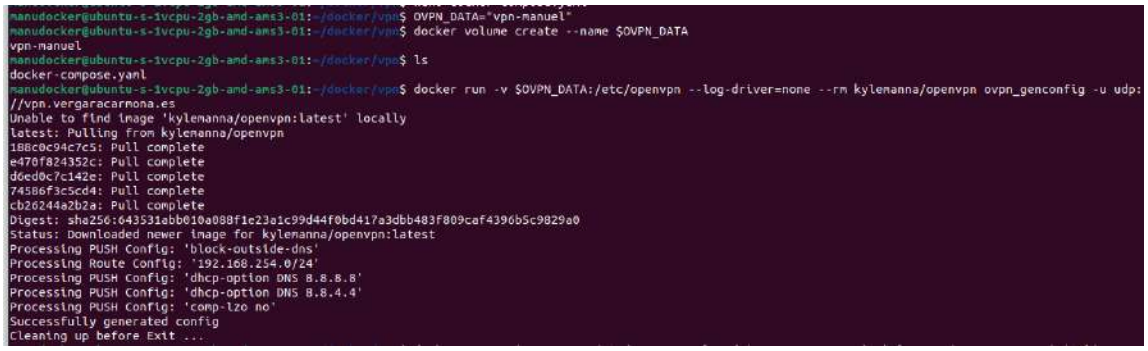
1. Crear la **DNS**

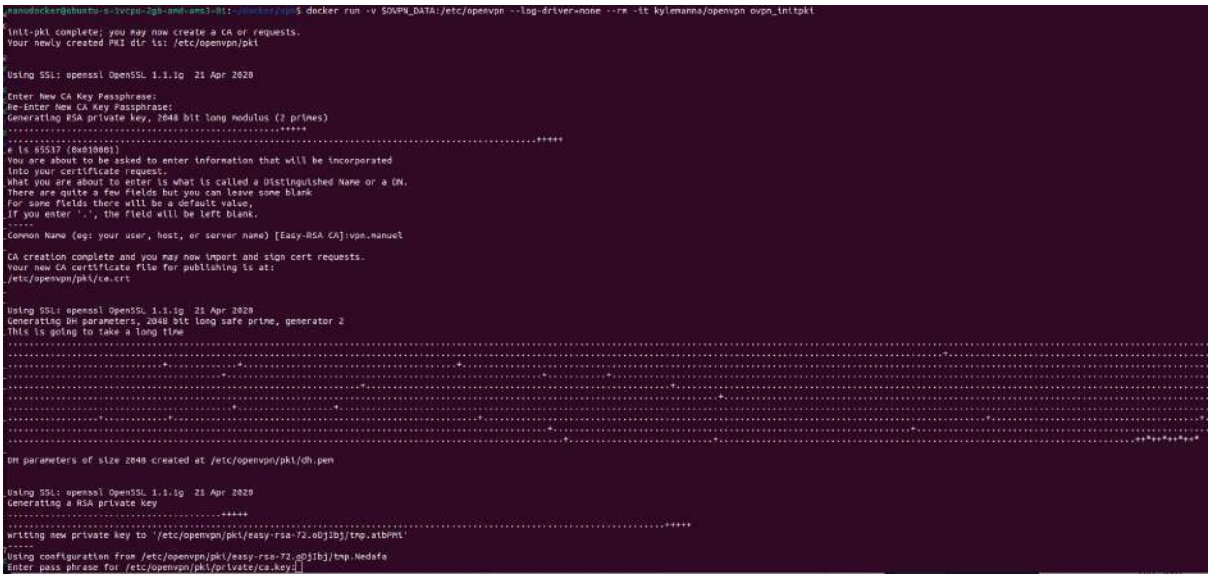
DNS records

Type	Hostname	Value	TTL (seconds)	
A	vpn.vergaracarmona.es	directs to 164.92.152.105	3600	More

2. Se **descarga la estructura** de directorios y ficheros con `git clone`
3. Se elige un **nombre para el volumen** de datos `$OVPN_DATA`.
   `OVPN_DATA="ovpn-data-example"`
4. Se **inicia el volumen** $OVPN_DATA que contendrá los archivos de configuración y los certificados.

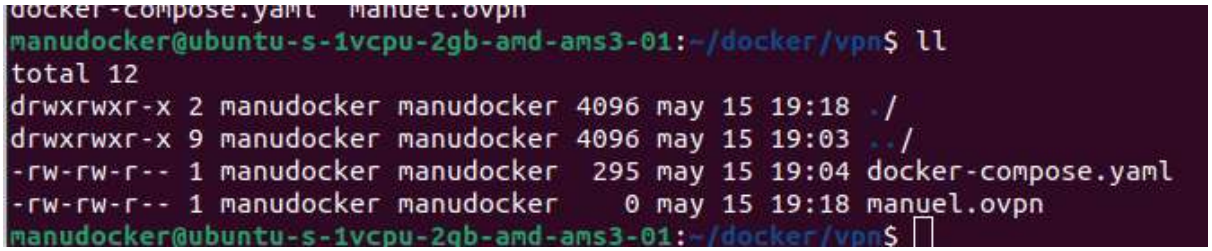
```
docker volume create --name $OVPN_DATA
docker run -v $OVPN_DATA:/etc/openvpn --log-driver=none --rm
kylemanna/openvpn ovpn_genconfig -u udp://vpn.vergaracarmona.es
docker run -v $OVPN_DATA:/etc/openvpn --log-driver=none --rm
-it kylemanna/openvpn ovpn_initpki
```





5. **Se inicia el proceso** del servidor OpenVPN
   `docker run -v $OVPN_DATA:/etc/openvpn -d -p 1194:1194/udp --cap-add=NET_ADMIN kylemanna/openvpn`
6. **Generar un certificado** de cliente sin frase de paso
   `docker run -v $OVPN_DATA:/etc/openvpn --log-driver=none --rm -it kylemanna/openvpn easyrsa build-client-full CLIENTNAME nopass`
7. **Recuperar la configuración del cliente** con los certificados incrustados
   `docker run -v $OVPN_DATA:/etc/openvpn --log-driver=none --rm kylemanna/openvpn ovpn_getclient CLIENTNAME > CLIENTNAME.ovpn`

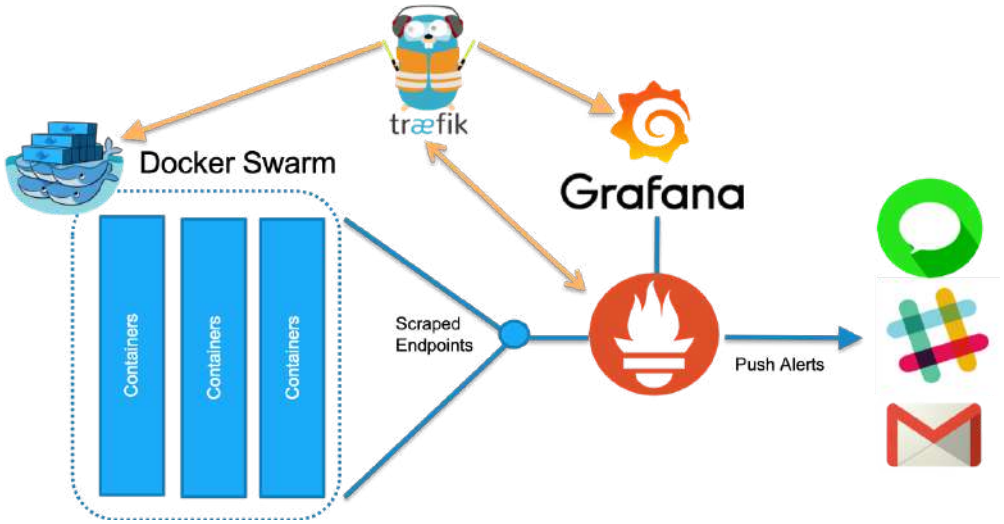
Al finalizar nos genera un documento que tendremos que descargar:



Lo ejecutamos con el siguiente comando:
`sudo openvpn manuel.ovpn`

https://github.com/stefanDeveloper/dodger/tree/main/openvpn

5.4.3. Herramienta de métricas



Junto con Traefik se configura **Prometheus** para la recolección de datos de series de tiempo y se despliega **Grafana** para la visualización.

Los pasos a seguir para el despliegue del servicio de métricas son:

1. Configurar las **DNS** adecuadas

DNS records

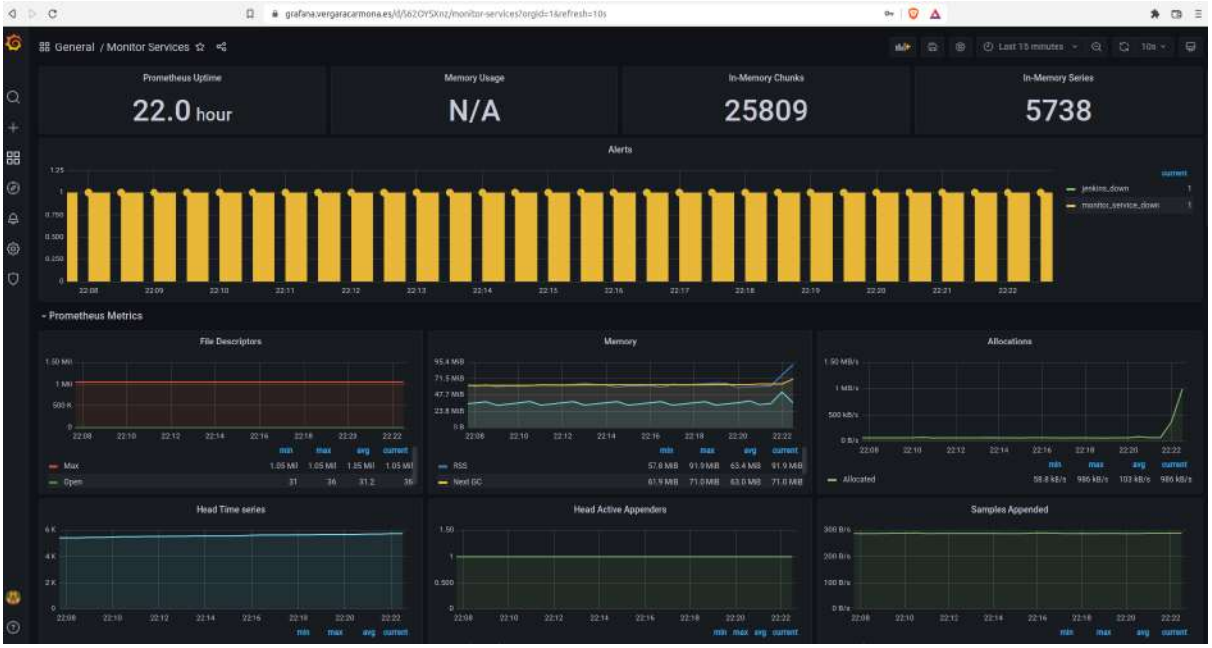
Type	Hostname	Value	TTL (seconds)	
A	prometheus.vergaracarmona.es	directs to 164.92.152.105	3600	More
A	grafana.vergaracarmona.es	directs to 164.92.152.105	3600	More

2. Descarga de la **estructura de archivos** y carpetas mediante `git clone`.
3. Adaptación del documento `docker-compose.yml` al servidor.
4. **Ejecutar la pila** con: `docker-composer up -d`

Estructura de documentos

```
manudocker@ubuntu-s-1vcpu-2gb-amd-ams3-01:~/docker/metrica$ tree
.
├── docker-compose.yml
├── grafana
│   ├── config.monitoring
│   └── provisioning
│       ├── dashboards
│       │   ├── dashboard.yml
│       │   └── traefik_rev4.json
│       └── datasources
│           └── datasource.yml
├── prometheus
│   ├── alert.rules
│   └── prometheus.yml
└── README.md

5 directories, 8 files
```

Al entrar en el enlace configurado se introduce el usuario. Ya hay **paneles configurados**.



5.4.4. CMS

Como hemos indicado, **Wordpress** es el CMS más usado que se hizo popular entre en el mundo bloguer. En él se construirá una simple landing page que servirá de página principal del proyecto y datos curriculares del autor.

En este caso, se utilizará el **dominio raíz** https://vergaracarmona.es Con lo cual, las DNS ya fueron configuradas con la configuración inicial del servidor.

Los pasos a seguir para el despliegue del wordpress son:

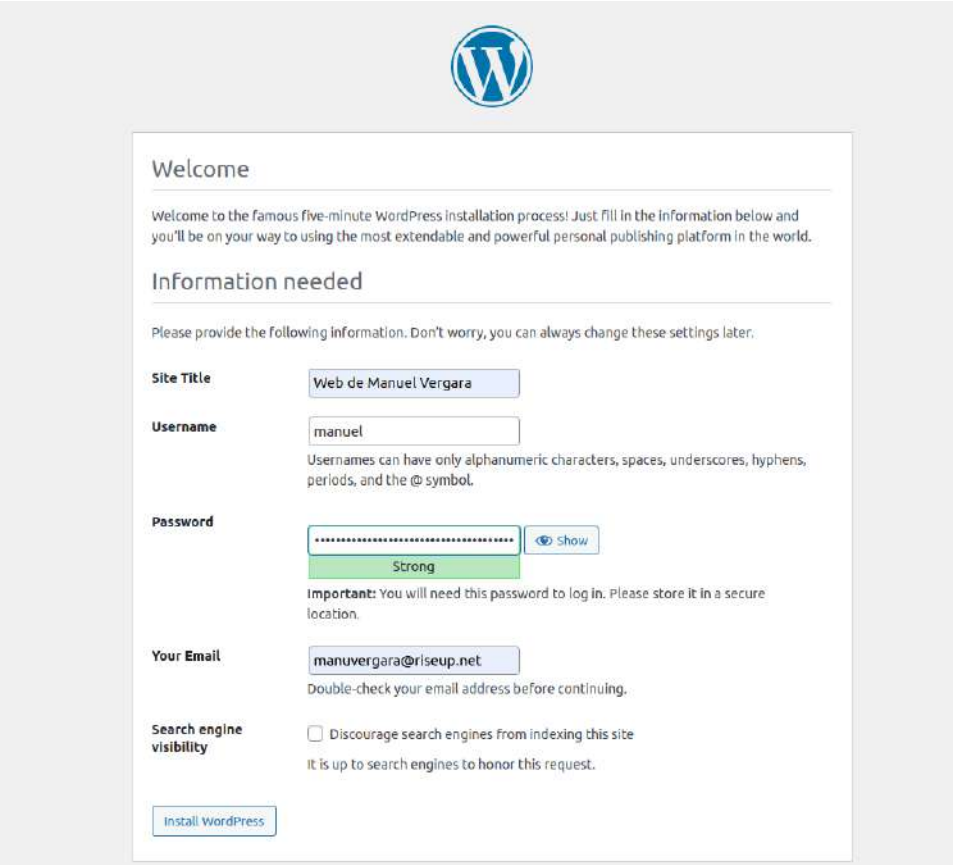
1. **Descarga de la estructura** de archivos y carpetas mediante git clone.
2. Adaptación del documento `docker-compose.yml` al servidor.
3. Dar los permisos adecuados a la estructura de directorios y ficheros públicos en **wp-content**.
4. **Ejecutar la pila** con: `docker-composer up -d`

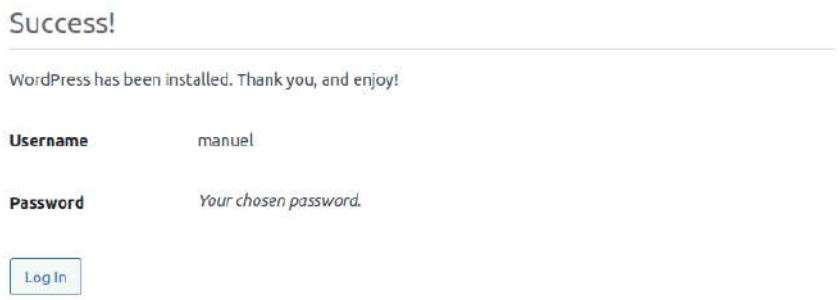
En este fichero **docker-compose** consta de tres contenedores. Por la parte **backend** la base de datos **mariadb** y, por la parte **frontend**, la webapp de **wordpress** y el gestor de BBDD **PHPmyAdmin** (https://pmawp.vergaracarmona.es/).

Estructura de datos:

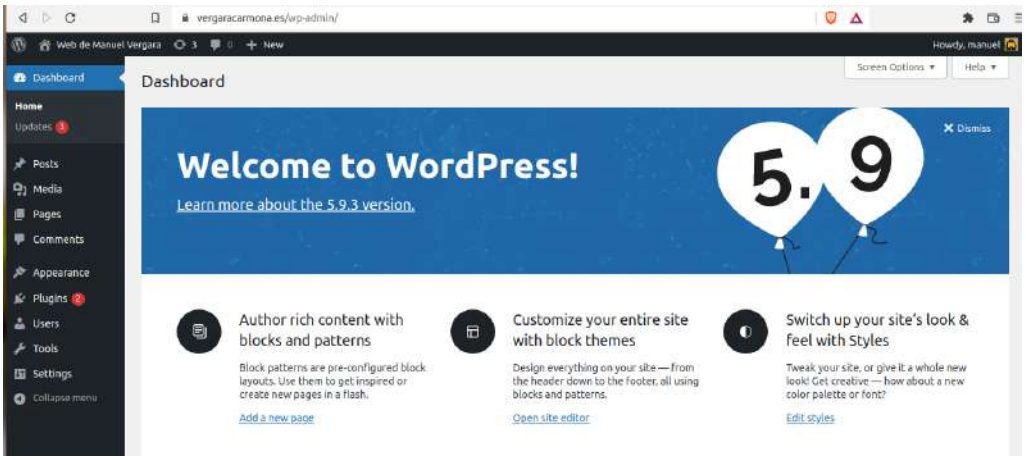
```
manudocker@ubuntu-s-1vcpu-2gb-amd-ams3-01:~$ tree -L 2 docker/wordpress/
docker/wordpress/
├── docker-compose.yml
├── docker-compose.yml.old
└── wp-data
    ├── index.php
    ├── license.txt
    ├── readme.html
    ├── wp-activate.php
    ├── wp-admin
    ├── wp-blog-header.php
    ├── wp-comments-post.php
    ├── wp-config-docker.php
    ├── wp-config.php
    ├── wp-config-sample.php
    ├── wp-content
    ├── wp-cron.php
    ├── wp-includes
    ├── wp-links-opml.php
    ├── wp-load.php
    ├── wp-login.php
    ├── wp-mail.php
    ├── wp-settings.php
    ├── wp-signup.php
    ├── wp-trackback.php
    └── xmlrpc.php
```

Iniciamos el CMS entrando en la url https://vergaracarmona.es/ para hacer algunas configuraciones extra





Después nos logueamos y ya estaremos en el escritorio backend de Wordpress



5.4.5. Servicio de nube

La herramienta **nextcloud** es de las más importantes para cubrir los servicios que recibe un smartphone. Esto se debe a la gran capacidad que tiene para integrar toda clase de herramienta a través de **plugins**, que dan nuevas funcionalidades como **calendario, tracker de gps, BBDD de contraseñas, RSS**, etc

Los pasos a seguir para el despliegue de Nextcloud son:

1. Configuración de los registros **DNS**.

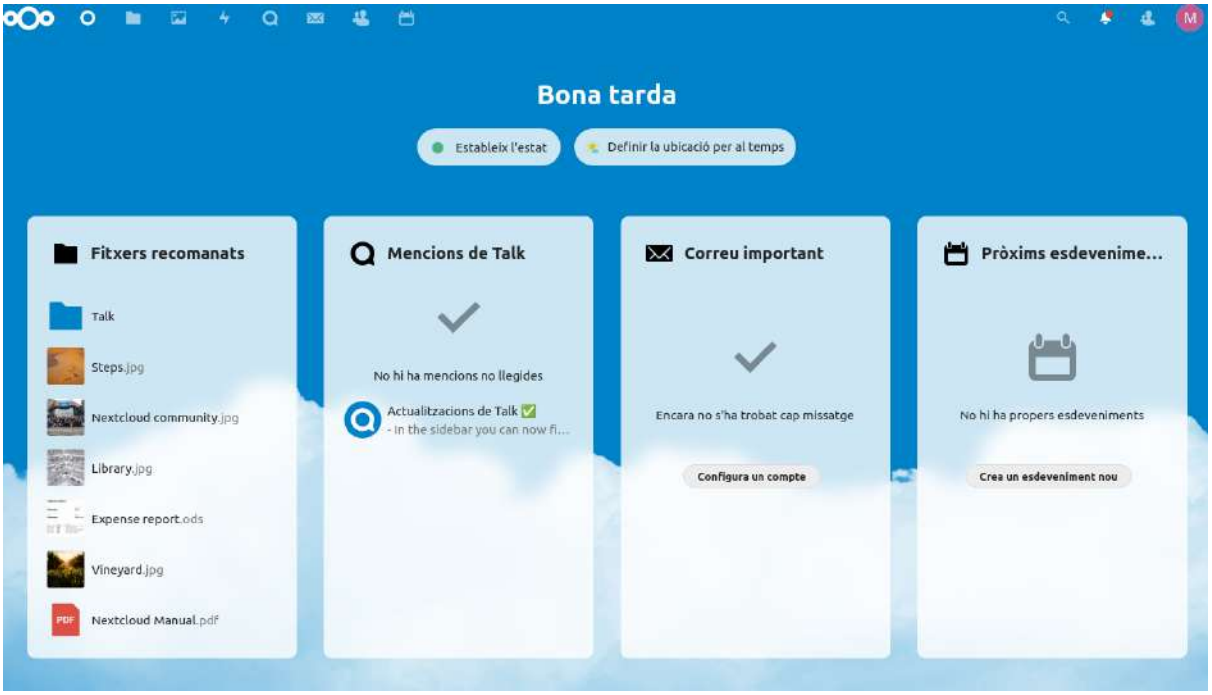
2. **Descarga de la estructura** de archivos y carpetas mediante `git clone`.
3. Adaptación del documento `docker-compose.yml` al servidor.
4. Adaptación del documento de variables `.env`
5. Dar los **permisos** adecuados a la estructura de directorios y ficheros públicos.
6. **Ejecutar la pila** con: `docker-composer up -d`

Estructura de directorios y ficheros

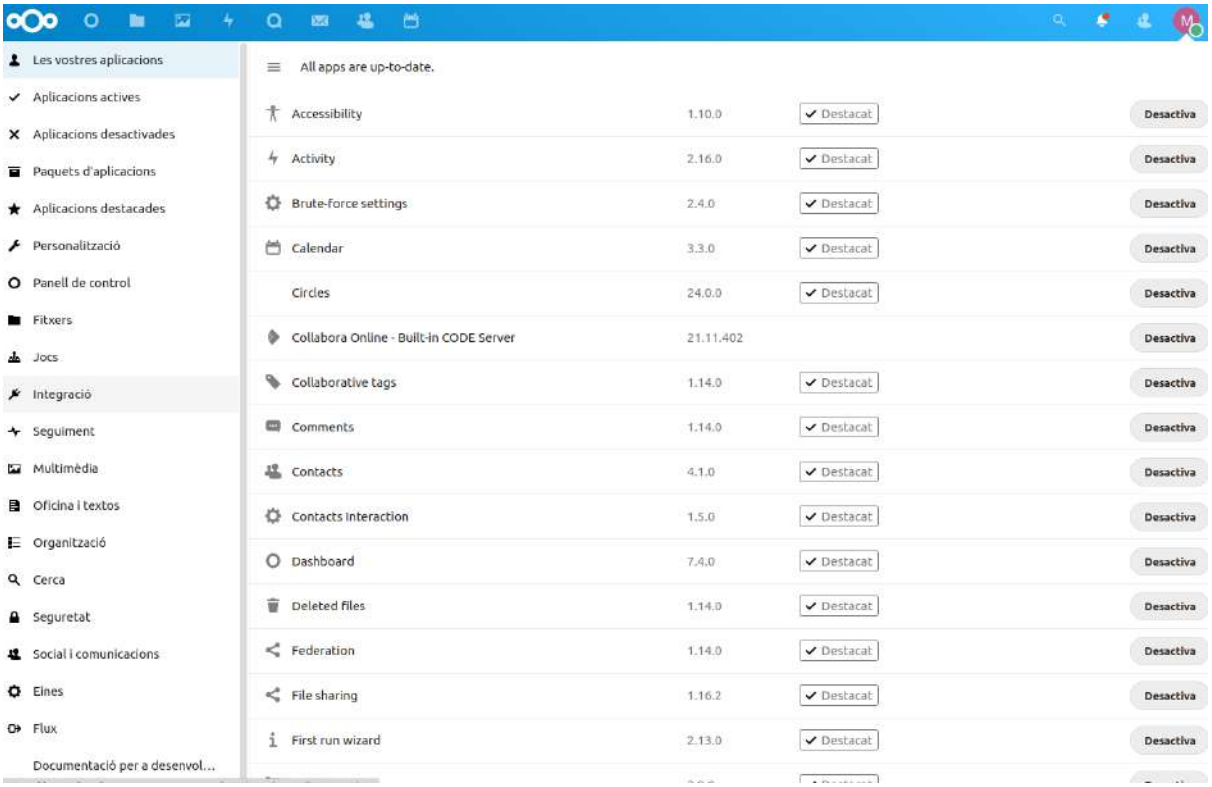
Iniciamos la webapp entrando en la url https://nextcloud.vergaracarmona.es/

Nos pide la contraseña configurada como usuario administrador y luego nos permite **instalar aplicaciones recomendadas**. Le doy a instalar.

Entonces nos lleva al **panel de control** de la aplicación



Para la **instalación de extensiones**, debemos entrar en aplicaciones (clicando en el avatar) y podremos elegir la deseada para descargar y/o activar/desactivar.



5.4.6. Mail Server

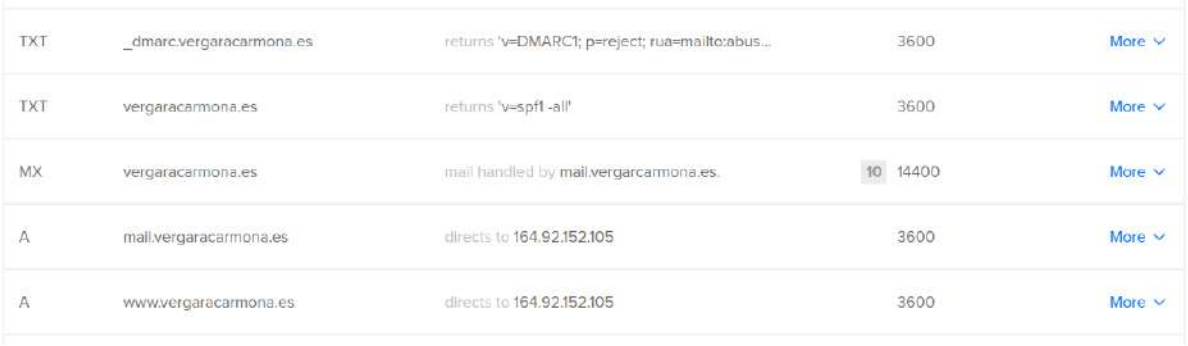
Para levantar un **servicio de mail seguro** se necesitan distintas herramientas:

- POP3, IMAP, SMTP con autenticación de usuario
- Aplicación de TLS
- Interfaz de correo web
- Filtrado de correo del lado del servidor, configuración de reglas a través de la interfaz web
- Filtro de spam y malware
- Utiliza RBL (listas de agujeros negros en tiempo real) para bloquear remitentes de spam ya conocidos
- Listas grises sólo cuando el correo entrante es probablemente spam
- Firma de mensajes DKIM
- Interfaz de gestión web para crear/eliminar cuentas, dominios y alias
- Soporte de cuentas de sólo envío que no pueden recibir pero sí enviar correos
- Los filtros IMAP, POP3 y de malware pueden desactivarse si no se utilizan
- Autocomprobación permanente mediante la función de comprobación de la salud de Docker
- Desarrollado con altos estándares de garantía de calidad
- Extensión de direcciones (-)

Este es un claro ejemplo de como docker facilita la instalación de servicios complejos.

Los pasos a seguir para el despliegue del mailserver son:

1. Configuración de las **DNS**



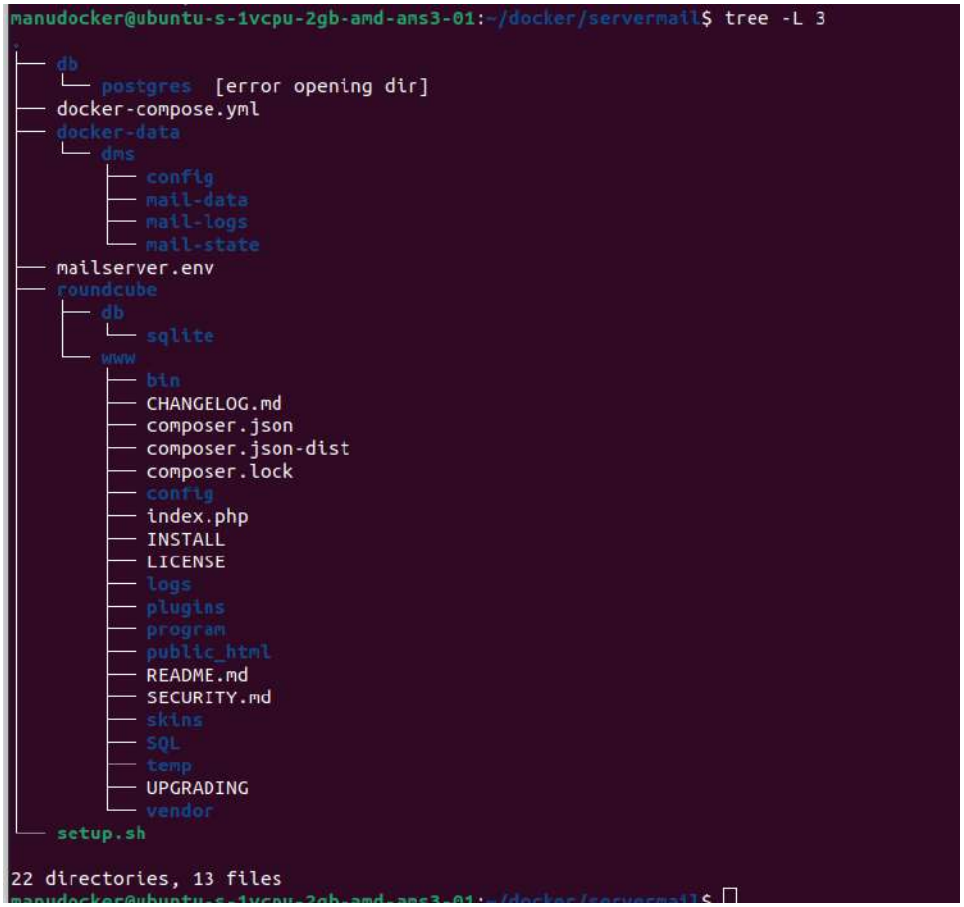
2. **Descarga de la estructura** de archivos y carpetas mediante `git clone`.
3. Adaptación del documento `docker-compose.yml`
4. Adaptación del documento de variables .env
5. En este caso, se utilizará un **script** para descargar las imágenes y dar la configuración adecuada. (`setup.sh`)

6. Después ya se podrá acceder a los servicios:

Servicio	Dirección
POP3 (starttls needed)	mail.vergaracarmona.es:110
POP3S	mail.vergaracarmona.es:995
IMAP (starttls needed)	mail.vergaracarmona.es:143
IMAPS	mail.vergaracarmona.es:993
SMTP	mail.vergaracarmona.es:25
Mail Submission (starttls needed)	mail.vergaracarmona.es:587
Management Interface	http://mail.vergaracarmona.es/manager/
Webmail	http://mail.vergaracarmona.es/webmail/
Rspamd Webinterface	http://mail.vergaracarmona.es/rspamd/

7. Para **crear un correo** se emplea el siguiente comando: `bin/production.sh run --rm web setup.sh`

Estructura de directorios y ficheros



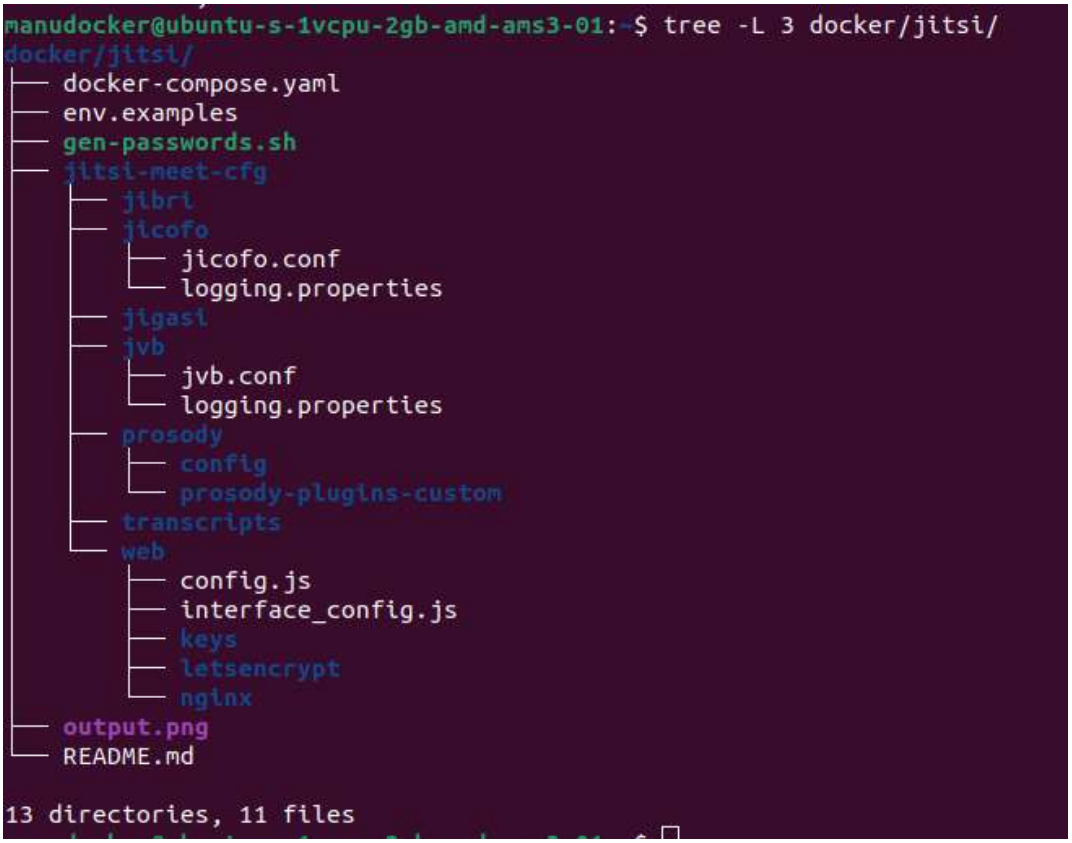
5.4.7. Servicio de videollamadas

Con el propio nextcloud se puede tener el servicio de **vídeollamadas** pero solo entre usuarios logueados. **Jitsi** permite crear salas públicas para hablar con cualquiera que tenga el enlace, es más estable y tiene mejor experiencia de usuario respecto a la configuración y el chat.

Para desplegar el servicio se deben seguir los siguientes pasos:

1. Configurar las **DNS** añadiendo el subdominio

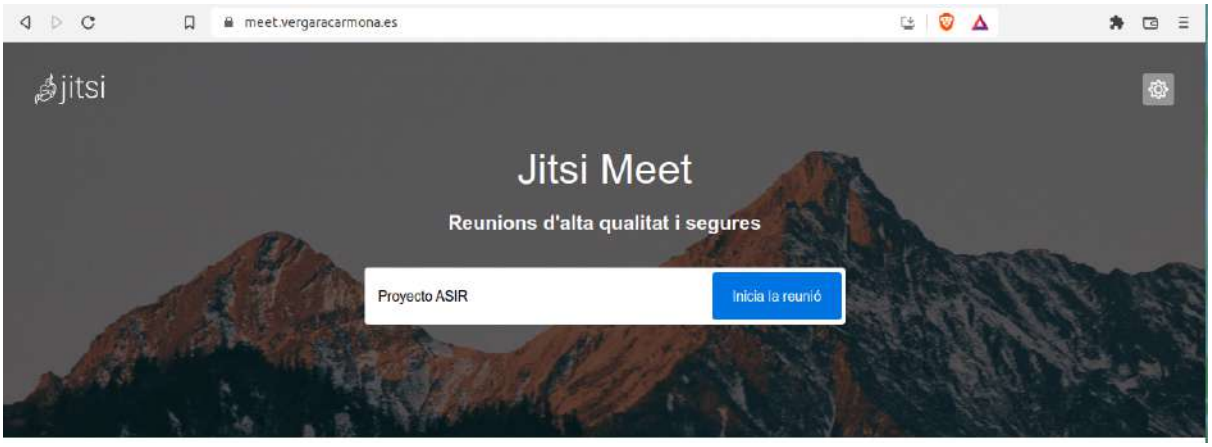
2. **Descargar la aplicación** con los documentos docker con el comando `git clone`. Estructura de los datos:



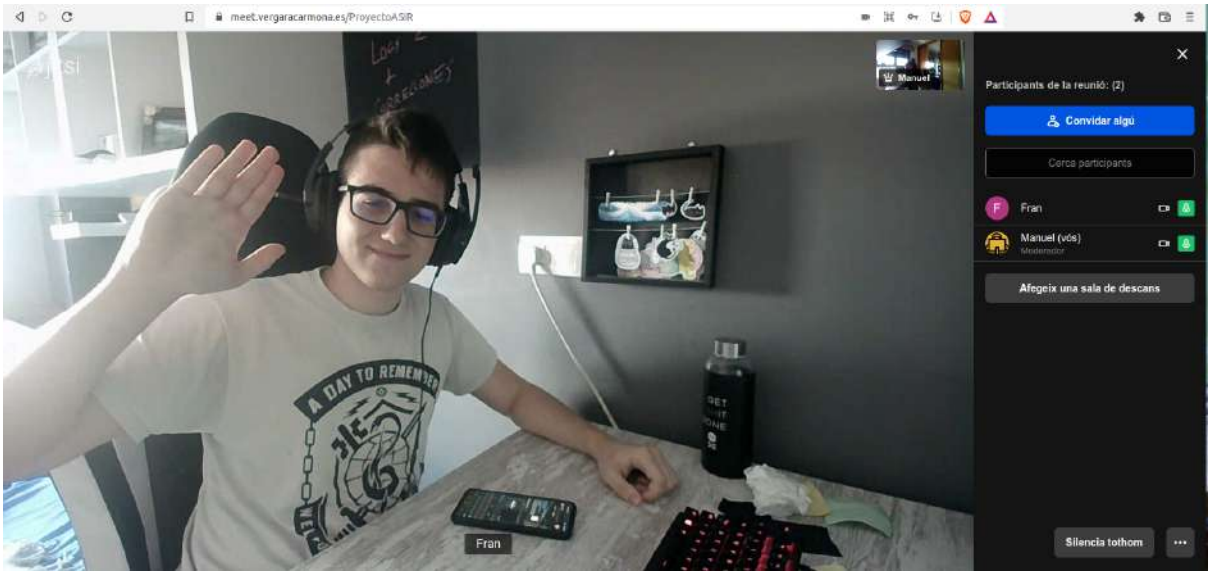
3. Configurar las **variables** adecuadas en el documento `.env`
4. Crear la **estructura** de carpetas necesarias. La mayoría ya se descarga con git
5. Configurar el archivo `docker-compose.yml` y adaptarlo a las necesidades.
6. Desplegar el contenedor con el comando `docker-compose up -d`
7. Crear las **credenciales** para entrar en la aplicación, desde el contenedor **prosody** con los siguientes comandos:

```
docker exec -it jitsi_prosody bash
prosodyctl --config /config/prosody.cfg.lua register usuariodeseado meet.jitsi contraseñadeseada
```

Ahora ya se puede crear la primera videoconferencia con la contraseña del anfitrión creada con prosody.

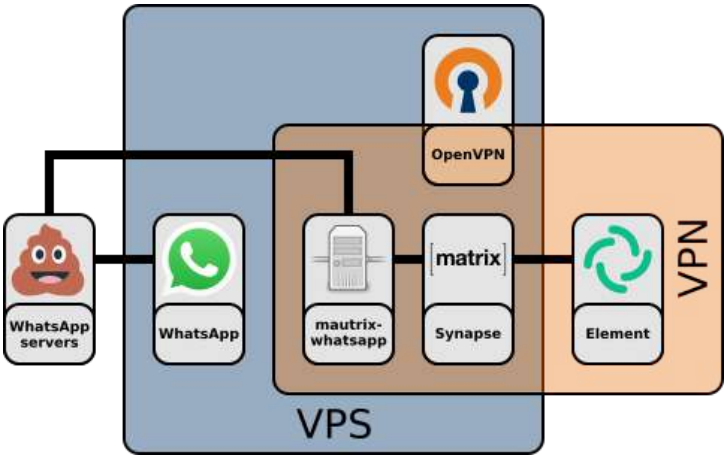


Prueba de llamada



5.4.8. Servicio de chats

Riot (ahora Element) es una aplicación de mensajería basada en **Matrix**. Busca reinventar el envío de mensajes: es **descentralizada, cifrada, sincronizada, cooperativa y gráfica.** Soportan grupos con cientos de usuarios y llamadas de audio y vídeo. Es un **sistema federado** que puede interpretar muchos tipos de mensajes. Por ejemplo:



Para desplegar el servicio se deben seguir los siguientes pasos:

1. Configurar las **DNS** añadiendo el subdominio

DNS records

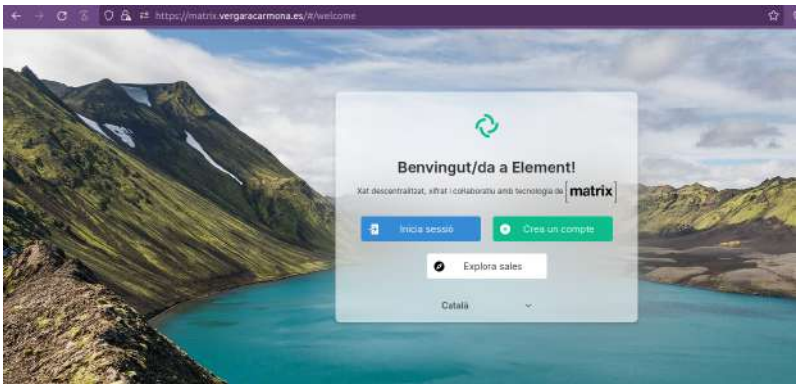
Type	Hostname	Value	TTL (seconds)	
A	riot.vergaracarmona.es	directs to 164.92.152.105	3600	More
A	matrix.vergaracarmona.es	directs to 164.92.152.105	3600	More

2. **Descargar la aplicación** con los documentos docker con el comando `git clone`. Estructura de los datos:

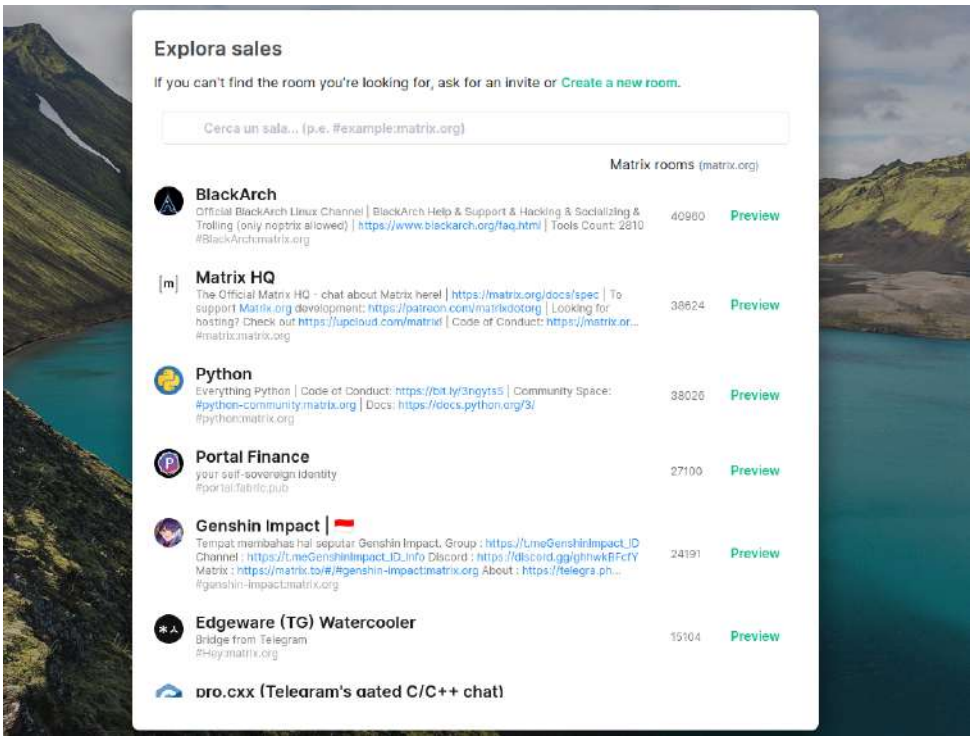
```
manudocker@ubuntu-s-1vcpu-2gb-amd-ams3-01:~/docker/matrix$ tree
.
├── data
│   ├── matrix
│   │   ├── nginx
│   │   │   ├── matrix.conf
│   │   │   └── www
│   │   ├── riot
│   │   │   └── config.json
│   │   └── synapse
│   │       ├── homeserver.yaml
│   │       └── media_store
│   └── postgres
│       └── data  [error opening dir]
└── docker-compose.yml

9 directories, 4 files
manudocker@ubuntu-s-1vcpu-2gb-amd-ams3-01:~/docker/matrix$
```

3. Configurar el archivo `docker-compose.yml` y adaptarlo al servidor.
4. **Desplegar el contenedor** con el comando `docker-compose up -d`

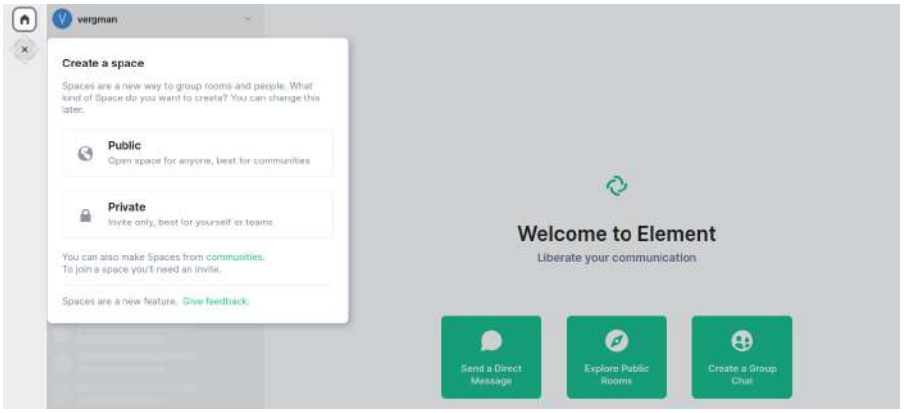
Cuando vamos a los subdominios riot o matrix vemos esto:



Podemos buscar **salas federadas**



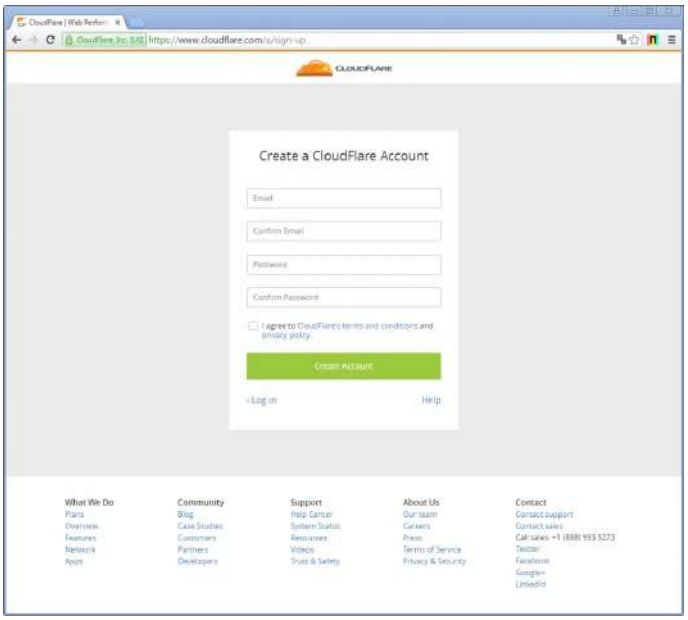
O podemos crear nuestros espacios



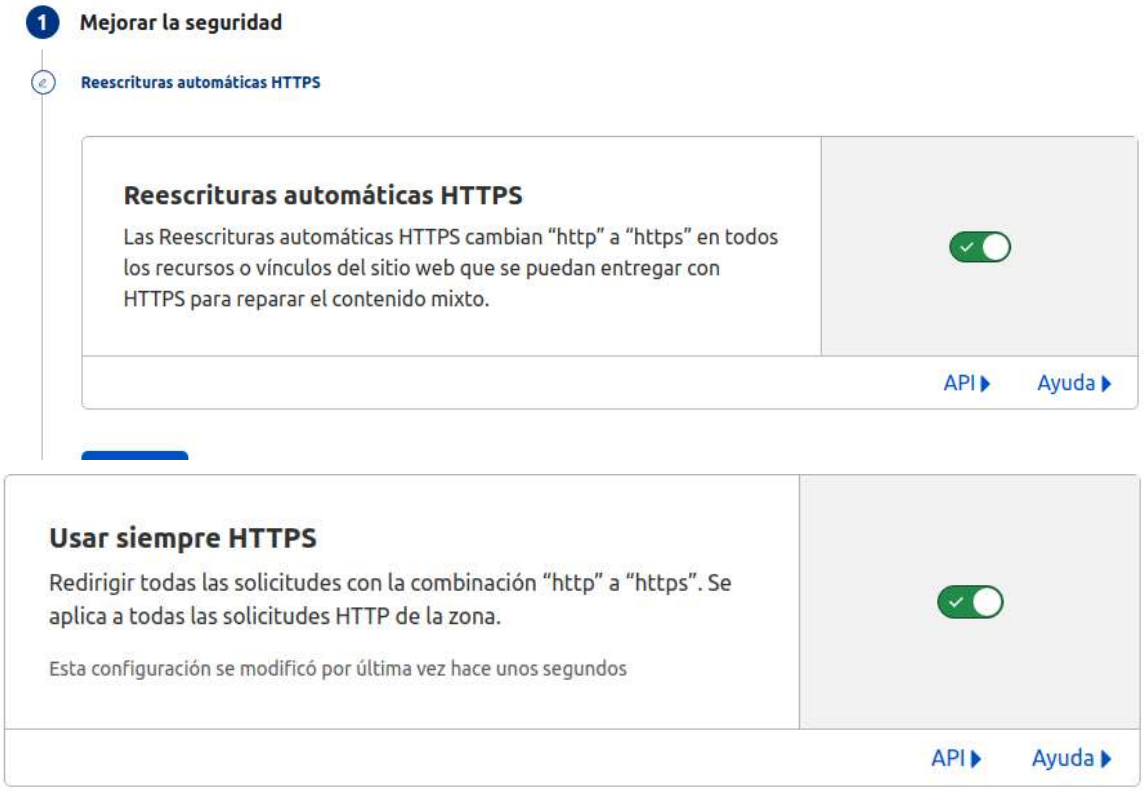
5.5. Conexión con red perimetral

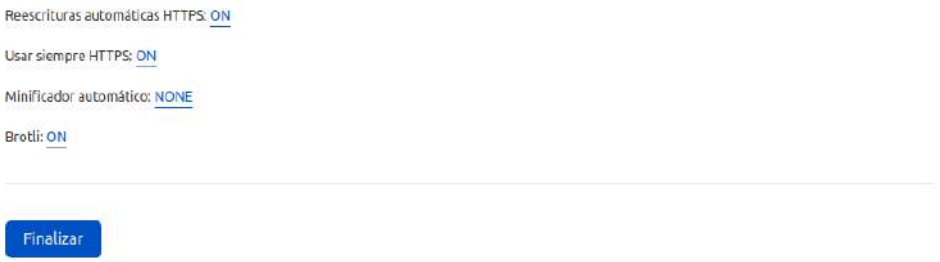
La red perimetral se configura a través de un servidor perimetral de **CDN** con la aplicación **Cloudflare**: https://dash.cloudflare.com/

Lo primero es **crear un usuario en la plataforma**

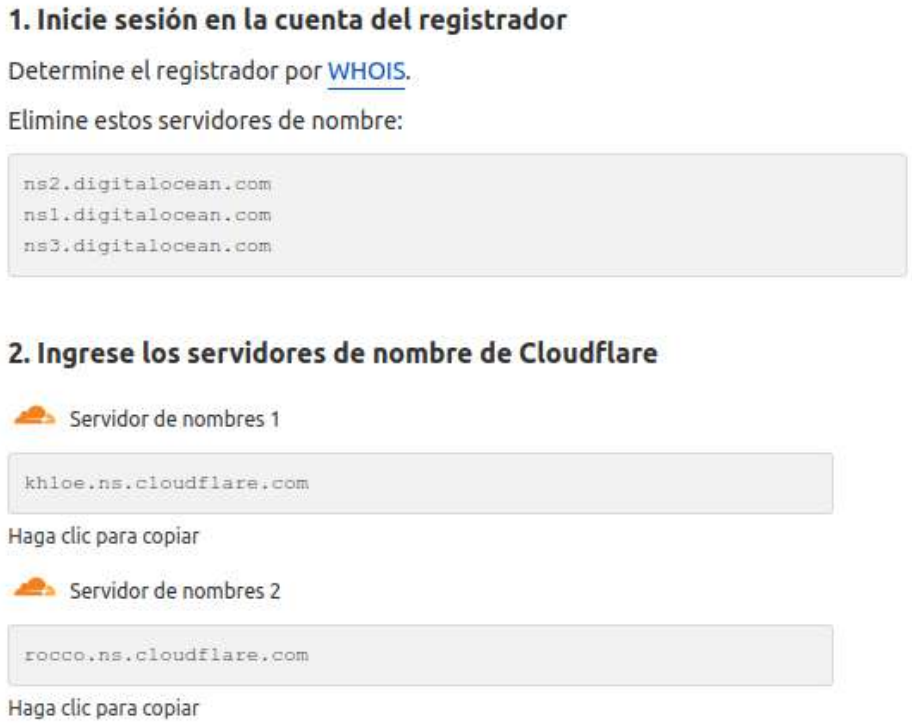


Una vez logueados, se escoge el plan gratuito y aparece una guía de inicio rápido. Acto seguido, analizará automáticamente las **DNS** del dominio. Aquí es mejor tener todas las DNS previstas configuradas en el servidor para que las tenga en cuenta desde el servicio.





Ahora tengo que **cambiar los registros NS** en DigitalOcean por los que me ofrecen aquí.



Una vez cambiados comenzará la **verificación**, que **puede tardar 24 horas**. A partir de entonces, CloudFlare pondrá una capa más de seguridad.

6. Conclusiones

Haciendo recopilación de lo vivido y, sobretodo, de los comentarios que me han hecho algunos profesionales de la informática, en apariencia, este proyecto parece farragoso en instalaciones y configuraciones. Pero nada más lejos, **una vez instalado y configurado el servidor base, se puede resumir en:**

1. Encontrar los servicios deseados.
2. Valorar técnicamente sus posibilidades.
3. Recopilar las imágenes docker de todos los factibles.
4. Desplegar los contenedores en producción.
5. Configurar cada servicio de manera independiente.

Siguiendo estos simples pasos el proyecto es muy asumible por cualquier usuario medio de la informática. Por mi parte, me he tenido que formar sobre todo en las herramientas **Docker y Git**. A día de hoy, **desplegar aplicaciones es simplemente correr un comando** con una imagen, sin necesidad de las tediosas horas de configuración de la era de los programas monolíticos.

Además, no se trata en este documento, pero existen **multitud de herramientas GUI** para poder desarrollar este mismo proyecto sin apenas tocar un shell. La visión gráfica podría convertir este proyecto en algo más popular, con interfaces intuitivas y técnicas drag & drop en lugar de líneas de comando. Sería un modo de **democratizar las aplicaciones Open Sources** del lado del servidor, acercando así al usuario final la posibilidad de dejar las dependencias tecnológicas, la obsolescencia programada y la minimización de la compatibilidad que han logrado instaurar los Big Tech para fidelizar a sus clientes.

6.1. Sobre el Open Source

Las herramientas privativas son un atraso al desarrollo tecnológico. La posibilidad de compartir, modificar y estudiar el código fuente lo ofrece el código abierto. **La copia es buena**, porque si un código se copia es porque es funcional, lo que no se copia acaba muriendo, por mucho que lo intenten mantener las empresas con actitud de la primera revolución industrial. Esta **vida propia del código abierto** en manos de desarrolladores dibuja a lo largo del tiempo **un esquema semejante a un árbol genealógico** como se puede apreciar en las distribuciones Linux.

Está claro que el Open Source no es la panacea. Por ejemplo:

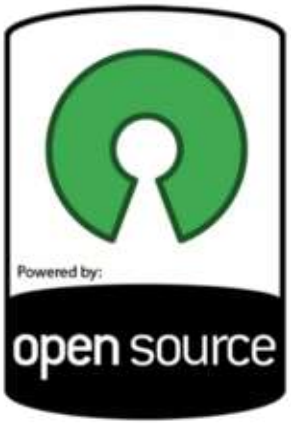
- Tiene una deuda pendiente con la **experiencia de usuario y el diseño**.
- Muchas veces **las actualizaciones se quedan estancadas** por muy práctica que sea la herramienta. Sobre todo cuando su funcionalidad se aleja de las necesidades de los desarrolladores (Véase las herramientas de diseño gráfico, software RRHH, contabilidad, etc).
- El punto anterior hace que este **muy denostado** por la supuesta calidad que da el soporte de una empresa.

Algo que parece acontecer en la actualidad es que los **Big Tech se están apropiándo del código abierto**. Lo compran, lo patentan y lo mezclan con código privativo o, lo que es peor, lo dejan morir porque podía ser un claro competidor en algún producto. Las prácticas empresariales no le sientan bien a la vida propia que pueda tener el código. Son herramientas para mejorar la vida de las personas y cubrir necesidades, pero como todo, se pueden utilizar para justo lo contrario.

Un ejemplo claro de apropiación del código es el seminario **Open Expo** dominado por las grandes tecnológicas. Normalmente se anuncian como la apertura del código de los productos empresariales para profesionales TI. Nada más lejos. Es un sitio más para la presentación de nuevos productos especializados para profesionales de TI.

Como última reflexión sobre el Open Source: **Un código abierto no implica que el software sea seguro ni que sea gratuito ni que sea libre**.

6.1.1. No es lo mismo “código abierto” que “software libre”



El software de **Código Abierto** tiene un significado cercano al **Software Libre**, aunque los dos términos no son idénticos. Ambas terminologías se refieren a un grupo similar de licencias y software, pero cada término alude a **diferentes ideologías**.

Según la **Free Software Fundation (FSF)**, para que una pieza de software se considere verdaderamente "libre", su licencia debe garantizar cuatro libertades esenciales a sus usuarios:

- **La libertad de ejecutar** el programa como se desee, para cualquier propósito.
- **La libertad de estudiar** cómo funciona el programa y cambiarlo para hacerlo como se desee. **El acceso al código fuente es una condición previa para esto.**
- **La libertad de redistribuir** copias del software.
- **La libertad de distribuir copias** de sus versiones **modificadas** a otros. Al hacer esto, puede darle a toda la comunidad la oportunidad de beneficiarse de los cambios. **El acceso al código fuente es una condición previa para esto.**

No todo el software de Código Abierto cumple al pie de la letra con estas definiciones.

La Open Source Initiative (OSI), la organización sin fines de lucro que apoya el desarrollo de software de Código Abierto, considera que el software debe cumplir con los siguientes criterios:

- **Redistribución gratuita** del software.
- El **código fuente** debe estar **disponible** públicamente.
- **El software se puede modificar y distribuir en un formato diferente** del software original.
- **El software no debe discriminar** a personas o grupos.
- **El software no debe restringir** el uso de otro software.

El Software Libre es más restrictivo porque no permite formatos diferentes en su redistribución y esto lo hace menos atractivo para las empresas.

6.2. Sobre los microservicios, Docker y los orquestadores de contenedores

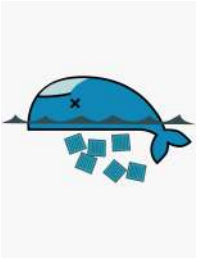
Con un **arquitectura de microservicios** como es el caso de docker, basada en contenedores de aplicación, nos encontramos con las siguientes características:

- **Se reduce el tamaño** de cada aplicación hasta puntos extremos, ya que tan solo contendrá lo necesario. Minimizan las dependencias optimizando su “scope”.
- Son fáciles de externalizar gracias a la **API REST**, así como altamente reutilizables.
- **Despliegue es más sencillo, ágil y rápido**. No se necesita seguir todos los pasos de instalación de cualquier servicio, tan solo es correr la imagen de ese servicio.
- **Maximizan su reutilización**.
- Los **ciclos de vida son independientes**.
- **Localización física abstracta**, tan solo necesitan un end-point para su gestión.
- **Evitan el mantenimiento** de los estados.
- Son **abordables por equipos pequeños** y desarrollables en poco tiempo.
- Permiten un **mantenimiento de la calidad y un testeo automático**.

Lo ideal sería tener **cada proceso distinto en un contenedor aislado**, sobre todo en los **casos de que se tratan cientos** de microservicios. El sistema está preparado para **reutilizar todos los procesos** sin necesidad de replicarlos.

Por contraste, **cuando los microservicios son pocos**, así como el personal de desarrollo, lo ideal es **concentrar los microservicios en contenedores** que contengan aplicaciones completas con sus distintos procesos. Es **un sistema monolítico aislado** en un contenedor de docker. Este sistema facilita la instalación así como el mantenimiento de la estructura lógica, haciendo una única discriminación: **Los servicios de backend y de frontend**.

Por ejemplo, en el caso del contenedor de wordpress contiene una LAMP, la webapp en sí de Wordpress y todas las dependencias necesarias del SO para poder funcionar. Pero está dividido en el Backend, con la BBDD, PHP y las dependencias necesarias del SO, y el frontend, con la aplicación web Wordpress y el servidor http.



Es obvio para cualquier Cloud Manager que **Docker es una tecnología del presente**, no del futuro. Quienes trabajamos en las tecnologías de la información sabemos que todo cambia muy rápido. La evolución de las herramientas y conceptos en la informática tienen una duración muy corta. Por ello, si la economía se conoce como la “ciencia miserable”, a **la informática se debería conocer como “disciplina condenada”**.

Los posibles sustitutos para Docker llegarán y darán un paso más en la evolución de los contenedores de aplicación. Se comenta una **combinación de los lenguajes de programación en uno**, incluido los archivos de configuración. Además, la **programación pilotada por IA, autoprogramación a tiempo real, programación mediante voz, sin lenguajes, a través de Machine Learning, distribuida por blockchain…** Queda mucho por ver y aunque docker esté dando un punto de inflexión está claro que **hay que aprovecharlo ahora** porque también desaparecerá.

6.3. Seguridad en la arquitectura

Cabe destacar que en el presente proyecto no se han implementado medidas complementarias de seguridad. Aun así, el **aislamiento de los servicios en contenedores** suele tener un componente adicional de seguridad debido a que los servicios se comunican entre ellos a través de redes locales, no obstante, es muy difícil desplegar una implementación por defecto en todas las arquitecturas ya que se deben adaptar a la topología lógica concreta que se esté construyendo.

Algunas **configuraciones de seguridad esenciales** que se pueden identificar como comunes en todo tipo de arquitectura son:

- Levantar los contenedores con **usuarios no-root**.
- **Parametrizar los contenedores entre ellos como si fuese una arquitectura monolítica** expuesta al exterior para añadir otra capa de seguridad.
- Desplegar la **última versión de software** y mantenerlo actualizado.
- Utilizar software, imágenes y todo tipo de documentos de configuración de **fuentes de confianza**, evitando así implementar malware en nuestra arquitectura.

Las **vulnerabilidades más comunes** en las arquitecturas de contenedores no son muy distintas a las arquitecturas monolíticas. Suelen ser:

- Una **mala configuración** de los servicios. Por ejemplo, exponer accidentalmente a internet un servicio con privilegios sin autenticación.
- Utilizar software de orquestación, máquina anfitriona o software de estructura de la arquitectura en una **versión sin parchear**. Es muy importante tener todos los servicios actualizados, sobre todo los que tienen contacto con el exterior.
- **Exponer a internet nodos de configuración innecesariamente**. Los servicios circundantes a la arquitectura que no sean necesarios desde el exterior no deben tener puertos expuestos para evitar ataques.
- **Tener habilitados los comandos en un contenedor expuesto al exterior**. Es imprescindible deshabilitar esta opción en todos los contenedores expuestos al exterior. Desde un contenedor que se pueda ejecutar comandos desde el exterior se puede tomar el control del host completo. En todo caso, una buena configuración puede evitar que un contenedor tenga privilegios ante el resto de contenedores.

Básicamente, siguiendo las **buenas prácticas de configuración** se consigue estrechar la superficie de vulnerabilidad. una web que nos puede ayudar a ello es **CIS benchmarks** (https://www.cisecurity.org/cis-benchmarks/), la cual contiene listas oficiales de las mejores prácitcas de seguridad para el software más conocido. **Estas guías son una referencia a seguir** para poder securizar servicios de internet, como por ejemplo SO, software de servidores, proveedores de cloud, dispositivos de red, etc

6.4. Control de los procesos y distribución de los servicios

En el presente proyecto, **tal como he ido añadiendo servicios los recursos del sistema se han ido agotando**. Un servicio en concreto, el servidor de mail, necesita todos los recursos del servidor actual para poder funcionar, cuando se desplegó se disparó la RAM hasta los límites, ralentizando el sistema y dejando los servicios impracticables, incluso llegando a dejarlo congelado.

Con todos los servicios arrancados se producen caídas en el servidor

En principio, el resto de servicios han podido convivir sin un uso masivo. En cualquier caso, se puede llegar a la conclusión que es mejor separar el servicio de mail del resto o escalar las características del servidor a lo que calculo que el mínimo viable y estable para una arquitectura como la planteada en este proyecto sería: 4 CPUs y 8 GB de RAM. Con lo que el precio sube a unos $40/mes.

Por el Almacenamiento no ha habido ningún problema, ya que docker consigue que se aprovechen los recursos ocupando un espacio poco significativo: 25 GB. El espacio contratado de 50 GB se quedaría pequeño en el caso de explotación del servicio de nube o ante un crecimiento de la web raíz mediante el blog, comentarios, etc.

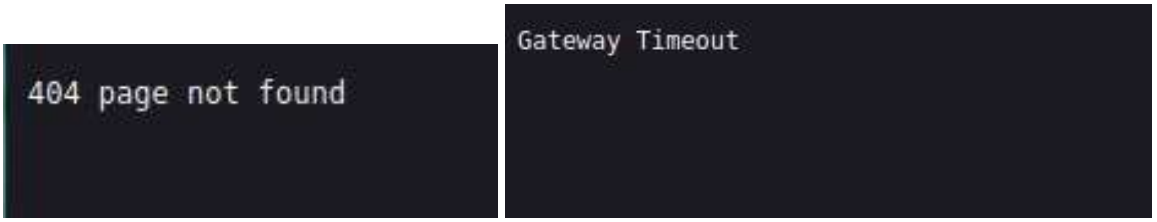
Este punto **supone un fracaso al planteamiento inicial** de tener un servidor asequible. Ya que encarece mucho el precio. Unas posibles soluciones serían que el servidor fuese compartido por una comunidad, una empresa o un grupo de al menos 3 o 4 personas; o, encontrar herramientas que cubran los mismos servicios pero necesiten menos recursos.

6.5. Incidencias al desplegar a producción

A la hora de desplegar las apps testeadas desde desarrollo a producción he tenido 2 problemas básicos:

- **Los permisos y la propiedad de los directorios** cambian.
- Las **limitaciones de los certificados SSL** autofirmados con Autoridad de Certificación let's encrypt.

Los problemas con los permisos/propiedad de las carpetas eran **debidos a que la MV en local tenía distintos usuarios y grupos**, con lo cual, cuando se descargan los archivos en el VPS los permisos cambiaban al usuario anfitrión o al root. En algunos casos los usuarios y grupos estaban dentro de los archivos de configuración de los servicios, lo que simplificaba la tarea. También si se especificaban mediante variables en los archivos de despliegue de docker.



Mensajes de respuesta de un servicio sin los permisos correctos.

La **solución ha sido en la mayoría de casos modificar los permisos/propiedades** a los ficheros/directorios (`chown` `chmod`) y, en los menos, instalar la aplicación desde cero. **Me queda la duda de cómo resolver esta transición**, ya que una de las ventajas de docker es su portabilidad sin tener porqué saber nada de la estructura total de las webapps o de las imágenes que transportan los contenedores.

Por otro lado, **la superación de las limitaciones de let's encrypt no tiene ninguna solución más que esperar** el tiempo estipulado por la Entidad de Certificación para poder comprobar de nuevo el certificado y que pase a ser confiable para el host..

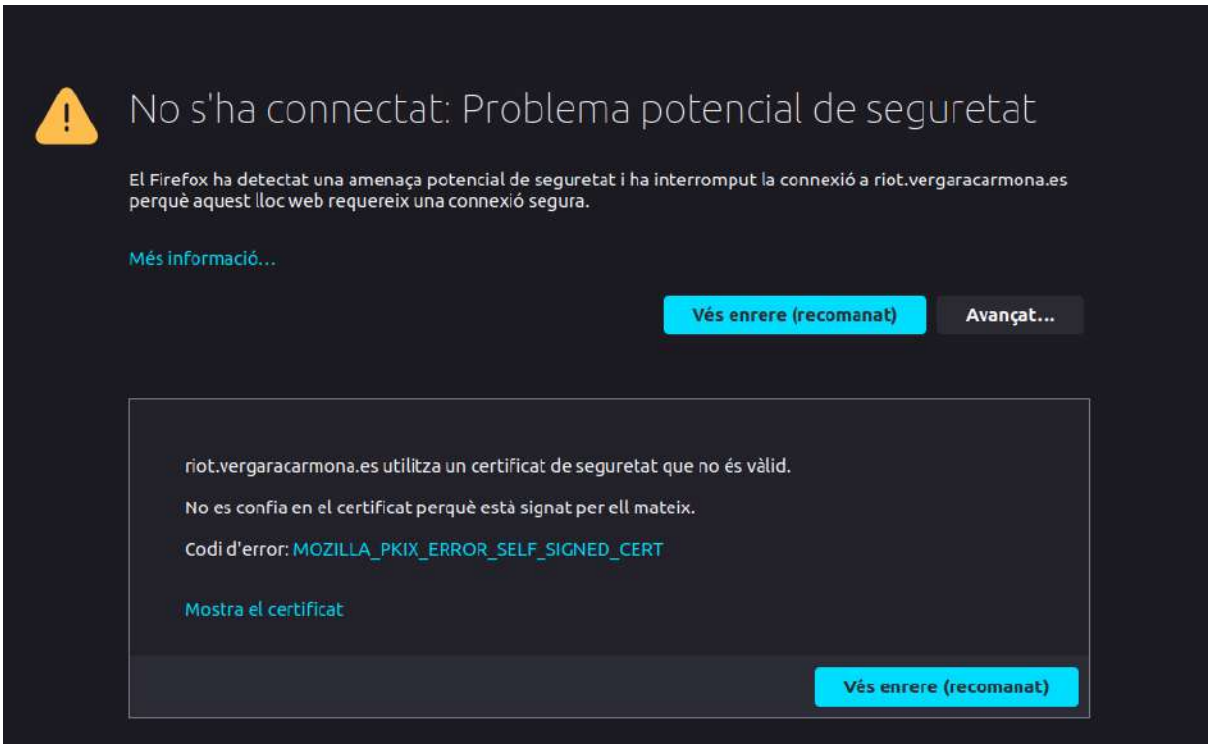
En concreto, las limitaciones que intuyó que se superó fueron las siguientes:

- El límite principal es Certificados por Dominio Registrado (50 por semana).
- Emitir 5 duplicados a la semana

La superación de las limitaciones **fue debida a un error por mi parte en el ciclo de cambios y comprobación**. Cada vez que hacía cambios en la construcción del docker-compose probaba sus resultados levantando el contenedor con `docker-compose up -d` y cuando veía el resultado (fuera positivo o negativo), volvía a bajarla con el comando `docker-compose down`. El error se debe a que cada vez que se desplegaba el contenedor **se volvía a crear un nuevo certificado** sin tener en cuenta el anterior, ya que son de autogeneración con Traefik. Así que en los primeros servicios no me afectó, pero en el momento que superé las limitaciones no podía entrar con el certificado SSL a los nuevos servicios que intenté instalar.

Fue un bloqueo que me costó entender y comprobar, porque no figuraba en ningún log.

Para evitar el error se puede usar docker-compose con `stop/start` o directamente con `restart`.



Resultado que devuelve el navegador cuando detecta que el certificado SSL no es confiable.

7. Detalles del software utilizado

Nombre	Versiones	Página oficial / Imagen Docker hub
Ubuntu server	20.04.4 LTS 22.04 LTS	https://ubuntu.com/download/server / https://hub.docker.com/_/ubuntu
VirtualBox	6.1	https://www.virtualbox.org/ / x
OpenVPN	2.4	https://openvpn.net/ / https://hub.docker.com/r/kylemanna/openvpn / https://gitea.vergaracarmona.es/manudocker/ProyectoASIR/src/branch/master/vpn
OpenSSH	9.0	https://www.openssh.com/ / x
Docker	Version: 20.10.14 API version: 1.41	https://www.docker.com/ / x
Swarm	1.2.9	https://github.com/docker-archive/classicswarm / x
Traefik	2.6.6	https://traefik.io/ / https://hub.docker.com/_/traefik / https://gitea.vergaracarmona.es/manudocker/ProyectoASIR/src/branch/master/traefik_portainer
Portainer	2.13.0	https://www.portainer.io/ / https://hub.docker.com/r/portainer/portainer / https://gitea.vergaracarmona.es/manudocker/ProyectoASIR/src/branch/master/traefik_portainer
Gitea	1.16.7	https://gitea.io/ / https://hub.docker.com/r/gitea/gitea / https://gitea.vergaracarmona.es/manudocker/ProyectoASIR/src/branch/master/gitea
Jitsi	7.0.0	https://jitsi.org/ / https://hub.docker.com/r/jitsi/web

Prosody	7.0.1	https://prosody.im/ / https://hub.docker.com/r/jitsi/prosody
Matrix	synapse 1.59.0	https://matrix.org/ / https://hub.docker.com/r/matrixdotorg/synapse / https://gitea.vergaracarmona.es/manudocker/ProyectoASIR/src/branch/master/matrix
Element	1.9.0	https://element.io/ / https://hub.docker.com/r/bubuntux/element-web
Postgres	10 11	https://www.postgresql.org/ / https://hub.docker.com/_/postgres
Redis	7.0.0	https://redis.io/ / https://hub.docker.com/_/redis
Prometheus	2.35.0	https://prometheus.io/ / https://hub.docker.com/r/bitnami/prometheus
Grafana	8.5.2	https://grafana.com/ / https://hub.docker.com/r/grafana/grafana
Nextcloud	24.0.0	https://nextcloud.com / https://hub.docker.com/_/nextcloud / https://gitea.vergaracarmona.es/manudocker/ProyectoASIR/src/branch/master/nextcloud
Wordpress	5.9.2	https://wordpress.org/ / https://hub.docker.com/_/wordpress / https://gitea.vergaracarmona.es/manudocker/ProyectoASIR/src/branch/master/wordpress
MariaDB	8.0	https://mariadb.org/ / https://hub.docker.com/_/mariadb
PHPmyAdmin	4.8.2	https://www.phpmyadmin.net/ / https://hub.docker.com/_/phpmyadmin / https://gitea.vergaracarmona.es/manudocker/ProyectoASIR/src/branch/master/wordpress

8. Webgrafía

La mayoría de fuentes de conocimientos se han encontrado en publicaciones de Internet. Pido disculpas a los autores originales si se ha producido algún error u omisión y ruego que se pongan en contacto conmigo para subsanar lo que esté en mi poder.

8.1. Manuales

- Documentación docker https://docs.docker.com/
- Documentación Github https://docs.github.com/es
- Documentación Kubernetes https://kubernetes.io/docs/home/
- Documentación Traefik https://doc.traefik.io/traefik/
- Documentación Portainer https://docs.portainer.io/v/ce-2.11/
- Documentación CAS https://apereo.github.io/cas/6.5.x/index.html
- Documentación Wordpress https://wordpress.org/support/
- Documentación Bitnami https://docs.bitnami.com/
- Documentación Nextcloud https://docs.nextcloud.com/
- Documentación Matrix https://docs.matrixx.com/
- Documentación BigBlueButton https://docs.bigbluebutton.org/
- Documentación OpenVPN https://openvpn.net/access-server-manual/tools-documentation-and-support/
- Documentación OpenSSH https://www.openssh.com/manual.html
- Documentación Jitsi https://jitsi.github.io/handbook/docs/intro/

8.2. Comunidades y foros

- Comunidad DigitalOcean https://www.digitalocean.com/community
- Github https://github.com/
- Stack overflow https://stackoverflow.com/
- Docker-hub https://hub.docker.com/
- Foro elhacker https://foro.elhacker.net/
- Foro coches, sección Electrónica e informática https://forococches.com/foro/forumdisplay.php?f=17
- Reddit https://www.reddit.com/

8.3. Guías

- Autoprotección digital contra la vigilancia: consejos, herramientas y guías para tener comunicaciones más seguras https://ssd.eff.org/es
- Guía de autodefensa del correo electrónico de la FSF https://emailselfdefense.fsf.org/es/
- The Ultimate Online Privacy Guide https://proprivacy.com/guides/the-ultimate-privacy-guide
- IVPN Privacy Guides https://www.ivpn.net/privacy-guides
- CIS benchmarks (https://www.cisecurity.org/cis-benchmarks/)

8.4. Libros

- "Software libre para una sociedad libre" Richard M. Stallman https://www.gnu.org/philosophy/fsfs/free_software2.es.pdf
- "La ética del hacker y el espíritu de la era de la información" Pekka Himanen http://eprints.rclis.org/12851/1/pekka.pdf
- "El enemigo conoce el sistema" Marta Peirano https://archive.org/details/el-enemigo-conoce-el-sistema-marta-peirano
- "Site reliability engineering" Betsy Beyer, Chris Jones, Jennifer Petoff y Niall Richard Murphy https://sre.google/sre-book/table-of-contents/
- “Microservices: The Journey So Far and Challenges Ahead” Pooyan Jamshidi, Claus Pahl, Nabor C. Mendonca, James Lewis y Stefan Tilkov. https://www.computer.org/csdl/magazine/so/2018/03/mso2018030024/13rRUNvgz2k

8.5. Vídeos

- Charla: "¿Por qué me vigilan, si no soy nadie?" Marta Peirano. TEDx Madrid https://archive.org/details/porquemevigilansinosoynadiemartapeiranotedxmadridnpe7i8wuupk
- Canal "Pelado Nerd" https://www.youtube.com/channel/UCrBzBOMcUVV8ryyAU_c6P5g

8.6. Repositorios de aplicaciones

- No more google https://nomoregoogle.com/
- Alternative to https://alternativeto.net
- Awesome humane tech https://github.com/humanetech-community/awesome-humane-tech
- Awesome selfhosted https://github.com/awesome-selfhosted/awesome-selfhosted
- Restore privacy https://restoreprivacy.com/
- Security in a box https://securityinabox.org/es/
- Prism break https://prism-break.org/en/
- Privacy tools https://www.privacytools.io/
- Privacy tools victorhck https://victorhck.gitlab.io/privacytools-es/
- Fleet linuxserver https://fleet.linuxserver.io/
- Docker stack including traefik, portainer, seafile, homer, openvpn, gitlab, wordpress, nextcloud, jenkins & Ansible https://github.com/stefanDeveloper/dodger
- ProyectoASIR https://gitea.vergaracarmona.es/manudocker/ProyectoASIR

8.7. Otros

- Wikipedia https://es.wikipedia.org/
- El taller del bit https://eltallerdelbit.com
- Rafrasenberg https://rafrasenberg.com/posts/
- Open Webinars https://openwebinars.net
- Microsiervos https://www.microsiervos.com/
- Diccionario informático https://techlib.net/index.html
- Blog de Red Hat https://www.redhat.com/es/blog
- Open Source - Red Hat https://opensource.com/
- Free Software Foundation https://www.fsf.org/es
- Free Software Foundation Europe https://fsfe.org/index.es.html
- Grupos de telegram de antiguos compañeros